The key
to the embedded world

LIN5200
Software Manual
Rev.105

Table of Contents

1. In This Manual

1.1 Terms and Conventions

Symbol/Tag	Description
	This symbol represents the handling of electrostatic - sensitive modules and/or components. These components are often damaged/destroyed with the transmission of a voltage higher than about 50V. Human body usually notices electrostatic discharges only above approximately 3,000V.
	This symbol indicates the possible use of voltages greater than 24V. Please note the relevant statutory regulations in this regard. Non-compliance with these regulations can lead to serious damage to your health and also cause damage/destruction of the component.
	This symbol indicates the possible source of danger. Acting against the procedure described can lead to possible damage to your health and/or cause damage/destruction of the material used.
	This symbol represents important details or aspects for working with TQ products.
Filename.ext	This specification is used to state the complete file name with its corresponding extension.
`Instructions / Examples`	Examples of application. e.g. • Specifying memory partitions • Processing a script • .....
Reference	Cross-reference to another section, figure or table.

1.2 Acronyms and Definitions

The following terminology and abbreviations are used:

Acronym	Full Form
BDM	Background Debug Mode
CPU	Central Processing Unit
CAN	Controller Area Network
CRT	Cathode Ray Tube
EEPROM	Electrically Erasable Programmable Read-Only Memory (Byte wise re-writable)
EMI / EMC	Electro Magnetic Interference / Electro Magnetic Compatibility
Flash	Electrically Erasable Programmable Read-Only Memory (Block Erase)
JTAG	Joint Test Action Group
MCU	Memory Control Unit
RTC	Real Time Clock
SDRAM	Synchronous Dynamic Random Access Memory
SMD	Surface Mounted Device

1.3 Tips on Safety

Improper or incorrect handling of the product can substantially reduce its life span.

1.4 Handling/ESD Tips

General Handling of your TQ Products

The handling and use of your TQ product may be done exclusively by qualified personnel.

Ensure that while using your TQ product, particularly while plugging in/out of modules, changing jumper settings, or connecting other external devices, the power supply is not connected to your TQ product. Violation of this guideline can result in damage/destruction of the module and cause danger to your health.

Improper handling of your TQ product would render the guarantee invalid.

Proper ESD handling

The ESD components must be used in workplaces which are apt for handling them in order to avoid damage or destruction.

1.5 Registered Trade Marks

The TQ-Components GmbH has striven to observe the copyright of the graphics and texts used in all publications, and to use those created by it, or those, which are license-free.

All brand names and trademarks contained in this publication are protected by copyright, unless specifically stated otherwise, by the corresponding owner or holder of the same. The sheer mention cannot be used to conclude that brand names and registered trademarks are not copy protected by third parties.

1.6 Imprint

TQ-Components GmbH
Schulstr. 29a
82234 Wessling

Tel: +49/(0)8153/9308-333
Fax: +49/(0)8153/9308-134

Email: info@tqc.de
Web: http://www.tq-components.com

1.7 Copyright

1.8 Disclaimer

The TQ-Components GmbH does not take over any guarantee for the currentness, correctness or quality of the information provided in this manual as also its further use. Claims lodged against TQ-Components GmbH, related to damages of material or intellectual nature, arising out of the use or non-use of information contained in this manual, or out of the use of incorrect or complete information, would not be entertained so long as there is no evidence of intentional or negligent fault on the part of TQ-Components GmbH.

The TQ Components GmbH reserves itself expressly to change or supplement contents of this manual or parts of it without prior intimation to this effect.

1.9 Revision History

Rev.	Date	Name	Pos.	Modification
001	01.12.04	MKR		Creation
002	13.04.05	ANW		Revision
003	19.05.05	MKR		Complete Update
004	05.08.05	MKR		Revision
005	04.11.05	MKR		Revision
101	15.11.05	ROE		First release
102	22.02.06	MKR		Add Java chapter
103	14.06.06	MKR		Generic Update, add descriptions for TB5200 board
104	13.07.06	MKR		Add description for TQM5200S, TQM5200 “Rev B”, high-boot option
105	23.05.07	MKR	7.4.1	Add chapter for PCAN driver for boards with MPC5200B cpu. Some additions for the TB5200 Rev. 300

2. Product Idea

U-Boot (Universal Bootloader) is the successor of PPCBoot (PowerPC Bootloader). This firmware serves as the bootloader for different operating systems and especially for Linux. The Open Source firmware U-Boot supports platforms based on embedded PowerPC as well as other CPU architectures. Its widespread use and support by the Open Source Community provides a wide base for development and ensures continuous development in the future.

Further information about the U-Boot project can be found under [6].

! note !	**U-Boot on modules of revision 10x:** The U-Boot version mentioned above is configured for the revision 200 of the TQM5200 module. This runs also on modules with revision 100. However, in this case, the I2C interface cannot be used (and thus, also an EEPROM that is available onboard). In order to use the I2C interface, the U-Boot must be compiled again. For this purpose, the option **`CONFIG_TQM5200_REV100`** is set in the file **`include/configs/TMQ5200.h`**.

! note !	**TQM5200 Module with 64MByte Flash Memory:** The 64 MByte Flash Memory is organized by the CPU in two Flash banks. The switchover takes place via a special bit in a CPU register. **The U-Boot, however, does not support this switchover currently.** A side effect is that the sectors 1-256 are reflected as sectors 257-512. When one accesses the sector 257, then physically the sector 1 is accessed! If physically, the sectors 257-512 are to be really used, then the Flash bank must be switched. Currently, this is implemented manually by setting the appropriate bit.

3. Licensing

The U-Boot bootloader and embedded Linux are subject to the GPL (GNU General Public License) [9]

4. First Steps

This chapter describes the first steps for setting up your STK52xx board and installing and running an embedded Linux on it.

This manual also could be used for the TB5200 (“Tinybox”). Where the usage of the TB5200 differs from the STK52xx additional descriptions are added to the chapters. For a detailed description of the TB5200 see the “TB5200 Hardware Manual”.

4.1 Connecting your Board

Use a serial Null Modem cable to connect the board with your PC. The cable needs two 9 pin DSUB female connectors. The data Lines TxD and RxD (Pin 2 and Pin 3) must be crossed internally. Other Pins are not used (except Pin 5: GND). Connect the cable with the lower connector of X2 (this is the connector besides the power supply connector, see also the hardware manual). The other end must be connected with your PC (e. g. COM1).

On your PC start a terminal program. You could use for example the provided Tera Term Pro. Tera Term Pro is a free terminal program for Windows. Configure the terminal program for the following settings:

- 115200, 8N1, no flow control

4.2 First Power-On

Connect your board to the power supply. On the serial terminal you should see the U-Boot boot messages:

```
U-Boot 1.1.3 (Apr 29 2005 - 09:42:02)

CPU:   MPC5200 v1.2 at 396 MHz
       Bus 132 MHz, IPB 132 MHz, PCI 66 MHz
Board: TQM5200 (TQ-Components GmbH)
       on a STK52XX baseboard
I2C:   85 kHz, ready
DRAM:  128 MB
FLASH: 64 MB
In:    serial
Out:   serial
Err:   serial
Net:   FEC ETHERNET
POST i2c PASSED
POST cpu PASSED
IDE:   Bus 0: not available
SRAM:  1 MB
VGA:   SMI501 (Voyager) with 8 MB
PS/2:  No device found
Kbd:   reset failed, no ACK

Type "run flash_nfs" to mount root filesystem over NFS

Hit any key to stop autoboot:  5
```

If you see the bootdelay counter, send any character over the serial line (by pressing a key in your terminal program) to stop U-Boot from autobooting. When autoboot is stopped, U-Boot enters the interactive command mode.

LIN5200.SWM.105.doc

4.3 Basic U-Boot Commands

This chapter is intended to give you a first starting point to the usage of U-Boot. It contains only a very brief description of some basic U-Boot commands. To take advantage of the many features of U-Boot it is strongly recommended to read the official U-Boot documentation: the DULG (DENX U-Boot and Linux Guide [1]) from DENX Software Engineering [5].

4.3.1 Informatory commands

4.3.1.1 coninfo – console information

The **coninfo** command prints information about the available console devices and the current assignment to stdin, stdout and stderr.

```
=> coninfo
List of available devices:
vga      80000002 S.O
serial   80000003 SIO stdin stdout stderr
nulldev  80000003 SIO
=>
```

Note: **coninfo** only shows information about currently available console devices. Thus, if there is for example no keyboard connected to the STK52xx, no “kbd” device is shown.

4.3.1.2 flinfo - FLASH information

The command **flinfo** prints information about the available FLASH memory. The information is queried directly from the CFI registers of the FLASH devices. The special Flags behind some Sector addresses give additional information about this sector: **RO**: this sector is read only; **E**: this sector is empty.

```
=> flinfo

Bank # 1: CFI conformant FLASH (32 x 16)  Size: 4 MB in 35 Sectors
 Erase timeout 16384 ms, write timeout 0 ms, buffer write timeout 1 ms, buffer s
ize 1
  Sector Start Addresses:
 FC000000  RO  FC008000  RO  FC00C000  RO  FC010000  RO  FC020000  RO
 FC040000  RO  FC060000  RO  FC080000  RO  FC0A0000      FC0C0000
 FC0E0000      FC100000      FC120000      FC140000      FC160000
 FC180000      FC1A0000 E    FC1C0000 E    FC1E0000 E    FC200000 E
 FC220000 E    FC240000 E    FC260000 E    FC280000 E    FC2A0000 E
 FC2C0000 E    FC2E0000 E    FC300000 E    FC320000 E    FC340000 E
 FC360000 E    FC380000 E    FC3A0000 E    FC3C0000 E    FC3E0000 E
```

4.3.1.3 help - print online help

The **help** command prints information about the specified command, or if no command is specified prints a list of available commands.

```
=> help help
help [command ...]
    - show help information (for 'command')
'help' prints online help for the monitor commands.

Without arguments, it prints a short usage message for all commands.

To get detailed help information for specific commands you can type
'help' with one or more command names as arguments.
```

4.3.2 Environment Variables

Environment variables are an important element of U-Boot. They are used to configure and change the behavior of U-Boot at run time and to pass information to the Linux kernel.

The environment variables are stored permanently in FLASH memory. At boot time they are copied from FLASH to RAM. Henceforth all operations with environment variables are done with the copies in RAM. To make changes permanent, the environment variables must be saved to FLASH with the **saveenv** command.

4.3.2.1 printenv - print environment variables

The **printenv** command prints the current value of the specified environment variable(s):

```
=> printenv bootdelay ethaddr
bootdelay=5
ethaddr=00:d0:93:03:e1:76
=>
```

If no environment variable is specified, a list with all environment variables is printed:

```
=> print
bootcmd=run net_nfs
bootdelay=5
baudrate=115200
preboot=echo;echo Type "run flash_nfs" to mount root filesystem over NFS;echo
netdev=eth0
rootpath=/opt/eldk/ppc_6xx
ramargs=setenv bootargs root=/dev/ram rw
nfsargs=setenv bootargs root=/dev/nfs rw nfsroot=$(serverip):$(rootpath)
addip=setenv bootargs $(bootargs) ip=$(ipaddr):$(serverip):$(gatewayip):$(netmas
k):$(hostname):$(netdev):off panic=1
flash_self=run ramargs addip;bootm $(kernel_addr) $(ramdisk_addr)
flash_nfs=run nfsargs addip;bootm $(kernel_addr)
net_nfs=tftp 200000 $(bootfile);run nfsargs addip;bootm
bootfile=/tftpboot/tqm5200/uImage
load=tftp 200000 $(u-boot)
u-boot=/tftpboot/tqm5200/u-boot.bin
update=protect off FC000000 FC05FFFF;erase FC000000 FC05FFFF;cp.b 200000 FC00000
0 $(filesize);protect on FC000000 FC05FFFF
ethact=FEC ETHERNET
ethaddr=00:d0:93:03:e1:76

Environment size: 833/65531 bytes
```

4.3.2.2 setenv - set environment variables

With the **setenv** command values could be assigned to environment variables. Please note that the variables are only set in RAM. To store it permanently the **saveenv** command must be used.

```
=> printenv bar
## Error: "bar" not defined
=> setenv bar This is a new example.
=> printenv bar
bar=This is a new example.
```

If no value is set, the specified environment variable is deleted (in RAM):

LIN5200.SWM.105.doc

```
=> printenv foo
foo=This is an example value
=> setenv foo
=> printenv foo
## Error: "foo" not defined
```

A '\' must be used to quote the characters '$' and ';', which are otherwise interpreted directly by U-Boot (as a kind of special commands).

```
=> setenv cons_opts console=tty0 console=ttyS0,\$(baudrate)
=> printenv cons_opts
cons_opts=console=tty0 console=ttyS0,$(baudrate)
```

4.3.2.3 saveenv - save environment variables

With the **saveenv** command the current states of **all** environment variables are saved to flash and thus stored permanently.

```
=> saveenv
Saving Environment to Flash...
Un-Protected 1 sectors
Un-Protected 1 sectors
Erasing Flash...
. done
Erased 1 sectors
Writing to Flash... done
Protected 1 sectors
Protected 1 sectors
```

4.4 Setting up a TFTP Server

Before you can transfer images to the target via Ethernet you must set up a TFTP server.

For a quick start you could use the provided TFTP server for Microsoft Windows on the TQ-CD: **tftpsrv.exe**.

Set the root directory of the TFTP server to a directory where you can store the images which are then made available for TFTP requests by the target.

```
C:\tftp>tftpsrv.exe p dc:\tftp\
TFTP Server V1.04
=================

Usage : tftpsrv [w] [p] [dRootDirectory]
        w    : enable write accesses
        p    : protocol server actions
        d??? : define a root directory

TFTP Server successful started
```

With the protocol option (**p**) enabled, all accessed to the TFTP server are logged:

```
RRQ received from 172.20.5.73 for c:\tftp\tftpsrv.exe
RRQ started, accessing c:\tftp\tftpsrv.exe
RRQ successful terminated, file c:\tftp\tftpsrv.exe
```

4.5 Network Configuration

Before the network commands could be used, the network has to be set up correctly. At least the following environment variables have to be set:

- ethaddr - ethernet address of the board (interface) -> this is preset by TQ
- ipaddr - IP address for the board (= target IP)
- serverip - IP address of your TFTP server

Your network environment may also require to set the following environment variables:

- netmask - subnet mask
- gatewayip - IP address of your gateway

Do not use the same IP address more than once in the same network. Save your environment after setting the variables to make the changes permanent.

```
=> set ipaddr 172.20.5.73
=> set serverip 172.20.5.13
=> save
Saving Environment to Flash...
Un-Protected 1 sectors
Un-Protected 1 sectors
Erasing Flash...
. done
Erased 1 sectors
Writing to Flash... done
Protected 1 sectors
Protected 1 sectors
```

4.6 Running a basic Linux System

In order to run Linux you must download a Linux kernel and a Linux root file system to the target. It is recommended to store both into flash.

4.6.1 Storing a Linux kernel to Flash

First you load the Linux kernel from your TFTP server to the RAM of your target board. This can be done with the **`tftp`** command:

```
=> help tftp
tftpboot [loadAddress] [bootfilename]
```

For a basic Linux system (which should run on all TQM5200 variants) use the image **uImage_tqm5200_2005-05-04_basic.** The kernel image must be copied to the exported root directory on your TFTP-Server. After that you can download the image to your target.

```
=> tftp 200000 uImage_tqm5200_2005-05-04_basic
Using FEC ETHERNET device
TFTP from server 172.20.5.13; our IP address is 172.20.5.73
Filename 'uImage_tqm5200_2005-05-04_basic'.
Load address: 0x200000
Loading: #################################################################
         #################################################################
         ###############################################################
done
Bytes transferred = 977879 (eebd7 hex)
=>
```

All numbers are interpreted by U-Boot as hex numbers**:**

After the TFTP transfer you can check the integrity of the image with the **iminfo** command (short: **imi**):

```
=> imi 200000

## Checking Image at 00200000 ...
   Image Name:   Linux-2.4.25
   Created:      2005-05-04  15:01:41 UTC
   Image Type:   PowerPC Linux Kernel Image (gzip compressed)
   Data Size:    977815 Bytes = 954.9 kB
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK
=>
```

If the flash sectors where the kernel image should be stored are not empty (this could be checked with the **flinfo** command), they must be erased before copying the kernel:

```
=> era fc0a0000 fc1fffff

.......... done
Erased 11 sectors
=>
```

With the **cp** command the image could be stored to flash:

```
=> cp.b 200000 fc0a0000 $(filesize)
Copy to Flash... done
=> imi fc0a0000

## Checking Image at fc0a0000 ...
   Image Name:   Linux-2.4.25
   Created:      2005-05-04  15:01:41 UTC
   Image Type:   PowerPC Linux Kernel Image (gzip compressed)
   Data Size:    977815 Bytes = 954.9 kB
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK
=>
```

The environment variable “filesize” is set automatically by the **tftp** command. Before command execution the term “$(filesize)” is replaced with the actual value of the variable “filesize”.

4.6.2 Storing a root File System to the Flash

Storing a root file system to the flash is similar to storing a Linux kernel to the flash:

1. Copy the image to the directory of the TFTP server
2. Load the image to the target RAM
3. Check the loaded image
4. Copy the image from RAM to flash

The used root file system image is a so called SELF (Simple Embedded Linux Framework) taken from the ELDK Installation:

```
=> tftp 200000 uRamdisk_SELF_ppx_82xx_eldk3.1_ppc
Using FEC ETHERNET device
TFTP from server 172.20.5.13; our IP address is 172.20.5.73
Filename 'uRamdisk_SELF_ppc_82xx_eldk3.1_ppc'.
Load address: 0x200000
Loading: #################################################################
         #################################################################
         #################################################################
         #################################################################
         ##################################################
done
Bytes transferred = 1581115 (18203b hex)
=> imi

## Checking Image at 00200000 ...
   Image Name:   Simple Embedded Linux Framework
   Created:      2004-11-10   9:45:02 UTC
   Image Type:   PowerPC Linux RAMDisk Image (gzip compressed)
   Data Size:    1581051 Bytes =  1.5 MB
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK
=> erase fc200000 fc3fffff

.................. done
Erased 16 sectors
=> cp.b 200000 fc200000 $(filesize)
Copy to Flash... done
=> imi fc200000

## Checking Image at fc200000 ...
   Image Name:   Simple Embedded Linux Framework
   Created:      2004-11-10   9:45:02 UTC
   Image Type:   PowerPC Linux RAMDisk Image (gzip compressed)
   Data Size:    1581051 Bytes =  1.5 MB
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK
=>
```

4.6.3 Booting Linux from Flash

After the Linux kernel and the root file system is stored to flash, the environment variable “bootargs” has to be set up with proper values. The content of the variable “bootargs” is passed to the Linux kernel as boot arguments (for example information about the root device or network configuration).

To simplify the setup work, there are some predefined environment variables which could be used to start Linux from flash:

```
=> print flash_self
flash_self=run ramargs addip;bootm $(kernel_addr) $(ramdisk_addr)
=> print ramargs
ramargs=setenv bootargs root=/dev/ram rw
=> print addip
addip=setenv bootargs $(bootargs) ip=$(ipaddr):$(serverip):$(gatewayip):$(netmas
k):$(hostname):$(netdev):off panic=1
=>
```

You only need to set the start addresses of the kernel image and the root file system:

```
=> set kernel_addr fc0a0000
=> set ramdisk_addr fc200000
=> save
```

```
Saving Environment to Flash...
Un-Protected 1 sectors
Un-Protected 1 sectors
Erasing Flash...
. done
Erased 1 sectors
Writing to Flash... done
Protected 1 sectors
Protected 1 sectors
=>
```

Now it's time to start a first Linux system:

```
=> run flash_self
## Booting image at fc0a0000 ...
   Image Name:   Linux-2.4.25
   Created:      2005-05-04  15:01:41 UTC
   Image Type:   PowerPC Linux Kernel Image (gzip compressed)
   Data Size:    977815 Bytes = 954.9 kB
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK
   Uncompressing Kernel Image ... OK
## Loading RAMDisk Image at fc200000 ...
   Image Name:   Simple Embedded Linux Framework
   Created:      2004-11-10   9:45:02 UTC
   Image Type:   PowerPC Linux RAMDisk Image (gzip compressed)
   Data Size:    1581051 Bytes =  1.5 MB
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK
   Loading Ramdisk to 00db8000, end 00f39ffb ... OK
Memory BAT mapping: BAT2=16Mb, BAT3=0Mb, residual: 0Mb
Linux version 2.4.25 (mkr@s020403) (gcc version 3.3.3 (DENX ELDK 3.1 3.3.3-8)) #
3 Wed May 4 17:01:06 CEST 2005
On node 0 totalpages: 4096
zone(0): 4096 pages.
zone(1): 0 pages.
zone(2): 0 pages.
Kernel command line: root=/dev/ram rw ip=172.20.5.73:172.20.5.13::::eth0:off pan
ic=1
Calibrating delay loop... 263.78 BogoMIPS
Memory: 11844k available (1660k kernel code, 548k data, 80k init, 0k highmem)
Dentry cache hash table entries: 2048 (order: 2, 16384 bytes)
Inode cache hash table entries: 1024 (order: 1, 8192 bytes)
Mount cache hash table entries: 512 (order: 0, 4096 bytes)
Buffer cache hash table entries: 1024 (order: 0, 4096 bytes)
Page-cache hash table entries: 4096 (order: 2, 16384 bytes)
POSIX conformance testing by UNIFIX
PCI: Probing PCI hardware
PCI: Cannot allocate resource region 0 of device 00:1a.0
Linux NET4.0 for Linux 2.4
Based upon Swansea University Computer Society NET3.039
Initializing RT netlink socket
Starting kswapd
Journalled Block Device driver loaded
Installing knfsd (copyright (C) 1996 okir@monad.swb.de).
JFFS2 version 2.2. (NAND) (C) 2001-2003 Red Hat, Inc.
PS/2 Multiplexer driver
Keyboard: not found
Mouse: not found
initialize_kbd: Keyboard reset failed, no ACK
ps2mult.c: keyboard command not acknowledged
Detected PS/2 Mouse Port.
```

```
pty: 256 Unix98 ptys configured
ttyS0 on PSC1
ttyS1 on PSC2
ttyS2 on PSC3
keyboard: Timeout - AT keyboard not present?(ed)
ps2mult.c: keyboard command not acknowledged
keyboard: Timeout - AT keyboard not present?(f4)
RAMDISK driver initialized: 16 RAM disks of 10240K size 1024 blocksize
loop: loaded (max 8 devices)
Uniform Multi-Platform E-IDE driver Revision: 7.00beta4-2.4
ide: Assuming 33MHz system bus speed for PIO modes; override with idebus=xx
Port Config is: 0x91550004
ipb=132MHz, set clock period to 7
GPIO config: 91550004
ATA invalid: 01000000
ATA hostcnf: 03000000
ATA pio1   : 100a0a00
ATA pio2   : 02040600
XLB Arb cnf: 8000a366
mpc5xxx_ide: Setting up IDE interface ide0...
Probing IDE interface ide0...
init_tqm5200_mtd: chip probing count 0
TQM5200-0: Found 2 x16 devices at 0x0 in 32-bit bank
 Amd/Fujitsu Extended Query Table at 0x0040
TQM5200-0: CFI does not contain boot bank location. Assuming top.
number of CFI chips: 1
cfi_cmdset_0002: Disabling erase-suspend-program due to code brokenness.
init_tqm5200_mtd: bank 1, name: TQM5200-0, size: 4194304 bytes
init_tqm5200_mtd: -22 command line partitions on bank 0
TQM5200 flash bank 0: Using static image partition definition
Creating 3 MTD partitions on "TQM5200-0":
0x00000000-0x000a0000 : "firmware"
0x000a0000-0x00200000 : "kernel"
0x00200000-0x00400000 : "initrd"
usb.c: registered new driver usbdevfs
usb.c: registered new driver hub
host/usb-ohci.c: USB OHCI at membase 0xf0001000, IRQ 44
host/usb-ohci.c: usb-0, Built-In ohci
usb.c: new USB bus registered, assigned bus number 1
hub.c: USB hub found
hub.c: 1 port detected
usb.c: registered new driver hiddev
usb.c: registered new driver hid
hid-core.c: v1.8.1 Andreas Gal, Vojtech Pavlik <vojtech@suse.cz>
hid-core.c: USB HID support drivers
mice: PS/2 mouse device common for all mice
NET4: Linux TCP/IP 1.0 for NET4.0
eth0: Phy @ 0x0, type LXT971 (0x001378e2)
IP Protocols: ICMP, UDP, TCP, IGMP
IP: routing cache hash table of 512 buckets, 4Kbytes
TCP: Hash tables configured (established 1024 bind 2048)
eth0: config: auto-negotiation on, 100FDX, 100HDX, 10FDX, 10HDX.
IP-Config: Guessing netmask 255.255.0.0
IP-Config: Complete:
      device=eth0, addr=172.20.5.73, mask=255.255.0.0, gw=255.255.255.255,
     host=172.20.5.73, domain=, nis-domain=(none),
     bootserver=172.20.5.13, rootserver=172.20.5.13, rootpath=
NET4: Unix domain sockets 1.0/SMP for Linux NET4.0.
RAMDISK: Compressed image found at block 0
Freeing initrd memory: 1543k freed
VFS: Mounted root (ext2 filesystem).
Freeing unused kernel memory: 80k init
eth0: status: link up, 100 Mbps Half Duplex, auto-negotiation complete.
```

```
BusyBox v0.60.5 (2004.11.10-09:40+0000) Built-in shell (msh)
Enter 'help' for a list of built-in commands.

# ### Application running ...

# cat /proc/version
Linux version 2.4.25 (mkr@s020403) (gcc version 3.3.3 (DENX ELDK 3.1 3.3.3-8)) #
3 Wed May 4 17:01:06 CEST 2005
```

4.6.4 Automatic start of Linux after Power-On

To make Linux start automatically after the power is switched on, the environment variable “bootcmd” has to be set. After the bootdelay counter has reached “0”, the command string stored in “bootcmd” is executed.

To make Linux boot from flash automatically set the “bootcmd” variable as follows:

```
=> set bootcmd run flash_self
=> save
Saving Environment to Flash...
Un-Protected 1 sectors
Un-Protected 1 sectors
Erasing Flash...
. done
Erased 1 sectors
Writing to Flash... done
Protected 1 sectors
Protected 1 sectors
=>
```

Now Linux is started automatically after a reset:

```
=> reset

U-Boot 1.1.3 (Apr 29 2005 - 09:42:02)

CPU:   MPC5200 v1.2 at 396 MHz
       Bus 132 MHz, IPB 132 MHz, PCI 66 MHz
Board: TQM5200 (TQ-Components GmbH)
       on a STK52XX baseboard
I2C:   85 kHz, ready
DRAM:  16 MB
FLASH:  4 MB
In:    serial
Out:   serial
Err:   serial
Net:   FEC ETHERNET
POST i2c PASSED
POST cpu PASSED
IDE:   Bus 0: not available
PS/2:  No device found
Kbd:   reset failed, no ACK

Type "run flash_nfs" to mount root filesystem over NFS

Hit any key to stop autoboot:  0
## Booting image at fc0a0000 ...
   Image Name:   Linux-2.4.25
   Created:      2005-05-04  15:01:41 UTC
   Image Type:   PowerPC Linux Kernel Image (gzip compressed)
   Data Size:    977815 Bytes = 954.9 kB
```

```
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK
   Uncompressing Kernel Image ... OK
## Loading RAMDisk Image at fc200000 ...
   Image Name:   Simple Embedded Linux Framework
   Created:      2004-11-10   9:45:02 UTC
   Image Type:   PowerPC Linux RAMDisk Image (gzip compressed)
   Data Size:    1581051 Bytes =  1.5 MB
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK
   Loading Ramdisk to 00db8000, end 00f39ffb ... OK
Memory BAT mapping: BAT2=16Mb, BAT3=0Mb, residual: 0Mb
Linux version 2.4.25 (mkr@s020403) (gcc version 3.3.3 (DENX ELDK 3.1 3.3.3-8)) #
3 Wed May 4 17:01:06 CEST 2005
…
```

5. Development Environment

If you want to do application development or configure the System for your needs, it is recommended to install the ELDK (Embedded Linux Development Kit) from DENX (www.denx.de).

The ELDK includes the GNU cross development tools, such as the compilers, binutils, gdb, etc., and a number of pre-built target tools and libraries necessary to provide some functionality on the target system.

It is recommended to use different ELDK versions for Linux Kernel 2.4 and 2.6:

- Linux Kernel 2.4: <= ELDK 3.1.1
- Linux Kernel 2.6: >= ELDK 4.0

5.1 Installing the ELDK

To Install the ELDK follow the instructions in the file **`README.html`** on the ELDK CD. As prerequisite you need a running Linux host. All familiar distributions like Red Hat, Fedora, SuSE, Debian and Ubuntu should do it.

Install the ELDK for the “ppc_6xx” architecture (= <cpu_family>), or install it for all ppc targets (with no argument for the **`install`** script).

A good source of information is again the DULG (http://www.denx.de/twiki/bin/view/DULG/ELDK).

5.2 Setting up a NFS Server

For a development environment it is very convenient when the host and the target can share the same files over the network. This could be achieved by the Network File System (NFS). The Linux host runs a NFS server and exports a directory that can be mounted from the target as the root file system.

To set up a NFS server on your Linux host, follow the documentation of your Linux distribution.

Add the directory that should be exported to the list of exported file systems, for instance in the file **`/etc/exports`**:

```
/opt/eldk/ppc_6xx/ *(rw,no_root_squash,sync)
```

This exports the directory **`/opt/eldk/ppc_6xx/`** with read and write permission to all hosts on the network (assuming that the installation directory of the ELDK was **`/opt/eldk`**).

5.3 Booting Linux over Network

Now you can boot Linux over network. For this the kernel is loaded from the TFTP server and the root file system is mounted via NFS from the Linux host.

For this purpose some environment variables are predefined:

```
=> print net_nfs
net_nfs=tftp 200000 $(bootfile);run nfsargs addip;bootm
=> print nfsargs
```

```
nfsargs=setenv bootargs root=/dev/nfs rw nfsroot=$(serverip):$(rootpath)
=> print addip
addip=setenv bootargs $(bootargs) ip=$(ipaddr):$(serverip):$(gatewayip):$(netmas
k):$(hostname):$(netdev):off panic=1
=>
```

Change the variables “bootfile” and “rootpath” to values according to your installation

```
=> set bootfile uImage_tqm5200_2005-05-04_basic
=> set rootpath /opt/eldk/ppx_6xx/
=> save
Saving Environment to Flash...
Un-Protected 1 sectors
Un-Protected 1 sectors
Erasing Flash...
. done
Erased 1 sectors
Writing to Flash... done
Protected 1 sectors
Protected 1 sectors
=>
```

Now Linux can be booted over network:

```
=> run net_nfs
Using FEC ETHERNET device
TFTP from server 172.20.5.13; our IP address is 172.20.5.73
Filename 'uImage_tqm5200_2005-05-04_basic'.
Load address: 0x200000
Loading: #################################################################
         #################################################################
         ################################################################
done
Bytes transferred = 977879 (eebd7 hex)
## Booting image at 00200000 ...
   Image Name:   Linux-2.4.25
   Created:      2005-05-04  15:01:41 UTC
   Image Type:   PowerPC Linux Kernel Image (gzip compressed)
   Data Size:    977815 Bytes = 954.9 kB
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK
   Uncompressing Kernel Image ... OK
Memory BAT mapping: BAT2=16Mb, BAT3=0Mb, residual: 0Mb
Linux version 2.4.25 (mkr@s020403) (gcc version 3.3.3 (DENX ELDK 3.1 3.3.3-8)) #
3 Wed May 4 17:01:06 CEST 2005
On node 0 totalpages: 4096
zone(0): 4096 pages.
zone(1): 0 pages.
zone(2): 0 pages.
Kernel command line: root=/dev/nfs rw nfsroot=172.20.5.13:/opt/eldk/ppc_6
xx/ ip=172.20.5.73:172.20.5.13::::eth0:off panic=1
Calibrating delay loop... 263.78 BogoMIPS
Memory: 13388k available (1660k kernel code, 548k data, 80k init, 0k highmem)
Dentry cache hash table entries: 2048 (order: 2, 16384 bytes)
Inode cache hash table entries: 1024 (order: 1, 8192 bytes)
Mount cache hash table entries: 512 (order: 0, 4096 bytes)
Buffer cache hash table entries: 1024 (order: 0, 4096 bytes)
Page-cache hash table entries: 4096 (order: 2, 16384 bytes)
POSIX conformance testing by UNIFIX
PCI: Probing PCI hardware
PCI: Cannot allocate resource region 0 of device 00:1a.0
Linux NET4.0 for Linux 2.4
Based upon Swansea University Computer Society NET3.039
```

```
Initializing RT netlink socket
Starting kswapd
Journalled Block Device driver loaded
Installing knfsd (copyright (C) 1996 okir@monad.swb.de).
JFFS2 version 2.2. (NAND) (C) 2001-2003 Red Hat, Inc.
PS/2 Multiplexer driver
Keyboard: not found
Mouse: not found
initialize_kbd: Keyboard reset failed, no ACK
ps2mult.c: keyboard command not acknowledged
Detected PS/2 Mouse Port.
pty: 256 Unix98 ptys configured
ttyS0 on PSC1
ttyS1 on PSC2
ttyS2 on PSC3
keyboard: Timeout - AT keyboard not present?(ed)
ps2mult.c: keyboard command not acknowledged
keyboard: Timeout - AT keyboard not present?(f4)
RAMDISK driver initialized: 16 RAM disks of 10240K size 1024 blocksize
loop: loaded (max 8 devices)
Uniform Multi-Platform E-IDE driver Revision: 7.00beta4-2.4
ide: Assuming 33MHz system bus speed for PIO modes; override with idebus=xx
Port Config is: 0x91550004
ipb=132MHz, set clock period to 7
GPIO config: 91550004
ATA invalid: 01000000
ATA hostcnf: 03000000
ATA pio1   : 100a0a00
ATA pio2   : 02040600
XLB Arb cnf: 8000a366
mpc5xxx_ide: Setting up IDE interface ide0...
Probing IDE interface ide0...
init_tqm5200_mtd: chip probing count 0
TQM5200-0: Found 2 x16 devices at 0x0 in 32-bit bank
 Amd/Fujitsu Extended Query Table at 0x0040
TQM5200-0: CFI does not contain boot bank location. Assuming top.
number of CFI chips: 1
cfi_cmdset_0002: Disabling erase-suspend-program due to code brokenness.
init_tqm5200_mtd: bank 1, name: TQM5200-0, size: 4194304 bytes
init_tqm5200_mtd: -22 command line partitions on bank 0
TQM5200 flash bank 0: Using static image partition definition
Creating 3 MTD partitions on "TQM5200-0":
0x00000000-0x000a0000 : "firmware"
0x000a0000-0x00200000 : "kernel"
0x00200000-0x00400000 : "initrd"
usb.c: registered new driver usbdevfs
usb.c: registered new driver hub
host/usb-ohci.c: USB OHCI at membase 0xf0001000, IRQ 44
host/usb-ohci.c: usb-0, Built-In ohci
usb.c: new USB bus registered, assigned bus number 1
hub.c: USB hub found
hub.c: 1 port detected
usb.c: registered new driver hiddev
usb.c: registered new driver hid
hid-core.c: v1.8.1 Andreas Gal, Vojtech Pavlik <vojtech@suse.cz>
hid-core.c: USB HID support drivers
mice: PS/2 mouse device common for all mice
NET4: Linux TCP/IP 1.0 for NET4.0
eth0: Phy @ 0x0, type LXT971 (0x001378e2)
IP Protocols: ICMP, UDP, TCP, IGMP
IP: routing cache hash table of 512 buckets, 4Kbytes
TCP: Hash tables configured (established 1024 bind 2048)
eth0: config: auto-negotiation on, 100FDX, 100HDX, 10FDX, 10HDX.
```

```
IP-Config: Guessing netmask 255.255.0.0
IP-Config: Complete:
      device=eth0, addr=172.20.5.73, mask=255.255.0.0, gw=255.255.255.255,
     host=172.20.5.73, domain=, nis-domain=(none),
     bootserver=172.20.5.13, rootserver=172.20.5.13, rootpath=
NET4: Unix domain sockets 1.0/SMP for Linux NET4.0.
Looking up port of RPC 100003/2 on 172.20.5.13
eth0: status: link up, 100 Mbps Half Duplex, auto-negotiation complete.
Looking up port of RPC 100005/1 on 172.20.5.13
VFS: Mounted root (nfs filesystem).
Freeing unused kernel memory: 80k init
INIT: version 2.84 booting
                Welcome to DENX Embedded Linux Environment
                Press 'I' to enter interactive startup.
Building the cache [  OK  ]
Mounting proc filesystem:  [  OK  ]
Configuring kernel parameters:  [  OK  ]
RTC_RD_TIME: Invalid argument
ioctl() to /dev/rtc to read the time failed.
Setting clock : Wed Dec 31 19:00:18 EST 1969 [  OK  ]
Setting hostname 172.20.5.73:  [  OK  ]
Mounting USB filesystem:  [  OK  ]
Activating swap partitions:  [  OK  ]
Finding module dependencies:  depmod: Can't open /lib/modules/2.4.25/modules.dep
 for writing
[FAILED]
Checking filesystems
[  OK  ]
Mounting local filesystems:  [  OK  ]
Enabling swap space:  [  OK  ]
modprobe: Can't open dependencies file /lib/modules/2.4.25/modules.dep (No such
file or directory)
INIT: Entering runlevel: 3
Entering non-interactive startup
Setting network parameters:  [  OK  ]
Bringing up loopback interface:  [  OK  ]
Starting system logger: [  OK  ]
Starting kernel logger: [  OK  ]
Initializing random number generator:  [  OK  ]
Starting portmapper: [  OK  ]
Mounting NFS filesystems:  [  OK  ]
Mounting other filesystems:  [  OK  ]
Starting xinetd: [  OK  ]

172 login: root
Last login: Wed Dec 31 19:00:34 on console
target# cat /proc/version
Linux version 2.4.25 (mkr@s020403) (gcc version 3.3.3 (DENX ELDK 3.1 3.3.3-8)) #
3 Wed May 4 17:01:06 CEST 2005
```

Changes you make to the exported root file system on your Linux host are visible directly on the root file system of the target. So you don't have to reboot your target system, if you want to change files on the target – change it on the Linux host and instantly they are visible on the target . This is, for instance, very comfortable if you are developing an application and often do a new compilation to check your modifications.

5.4 Building the Linux Kernel

If you want to adjust the Linux kernel to your demands, for instance to reduce memory footprint, or to support a special file system, or to support a special hardware, you had to compile your own Linux kernel. With the ELDK it is very easy to build the Linux kernel from the sources.

5.4.1 Installing the Sources

Before you could build your own Linux kernel, you have to install the sources on your Linux host. A good place to do this is your home directory. You don't need root rights to install and build a Linux kernel for the target.

5.4.1.1 Installing from a tar Archive

The easiest way to install the Linux sources on your host is to use a tar archive. For example **linuxppc_2_4_devel_2005-04-14_tq.tar.gz** from the TQ-CD:

```
# cd <your homedir>
# cp <path on cd>/linuxppc_2_4_devel_2005-04-14_tq.tar.gz ./
# tar -xzvf linuxppc_2_4_devel_2005-04-14_tq.tar.gz
linuxppc_2_4_devel/
linuxppc_2_4_devel/fs/
linuxppc_2_4_devel/fs/CVS/
…
```

5.4.1.2 Installing from DENX git archive

"git" is a "directory content manager" used and developed by Linus Torvalds and the Linux Community for the Linux kernel. The formerly used CVS was superseded completely by git now (at least for Linux and U-Boot).

You can browse the public DENX git repositories with a web browser (see [10]).

Or you could clone the git archive to your local host. This requires that "git" and "cogito" is installed on your host.

Clone the Linux git archive via git protocol:

```
#cg-clone git://source.denx.net/git/linuxppc_2_4_devel.git <installdir>

…
```

Hint: If you are sitting behind a firewall, make sure that the "git" port (port 9418) is not blocked.

Alternatively you could clone the Linux git archive via http:

```
#cg-clone http://source.denx.net/git/linuxppc_2_4_devel.git <installdir>
…
```

5.4.1.3 Where could I get recent sources?

The source tar archives on the TQ-CD are a good starting point for playing with Linux on the TQM5200, but they are probably not the most recent sources available.

For current sources please use the DENX git archive (as described above).

Because it may take some time till changes made it into the DENX git archive, please contact TQ-Components directly for the latest sources or patches available (website: http://www.tqc.de, or via email or telephone).

5.4.2 Building Linux with a default Configuration

Make sure, that ELDK is installed and the environment variables $PATH and $CROSS_COMPILE are set up properly. $PATH must contain the path where you installed ELDK.

```
# echo $PATH
/opt/eldk/usr/bin:/opt/eldk/bin:/sbin:/usr/sbin:/bin:/usr/bin:/usr/X11R6/bin:/usr/loc
al/bin
# echo $CROSS_COMPILE
ppx_6xx-
```

Change into the root directory of the Linux sources and do a complete cleanup of the sources (just to be sure):

```
# cd linuxppc_2_4_devel
# make mrproper
```

Then copy the predefined Kernel configuration file to the top of the source directory. This configuration is suitable for all TQM5200 variants. Because not all module contain a graphic controller, no graphic driver is configured.

```
# cp .config_tqm5200_2005-05-04_basic .config
```

With the next steps some include files are generated automatically and the dependencies for the make process are created.

```
# make oldconfig
…
# make dep
…
```

With the dependencies built, we can now compile the kernel image (according to your system performance this could take between 5 minutes and half an hour):

```
# make uImage
…
ppx_6xx-objcopy --strip-all -S -O binary /home/fpe/linuxppc_2_4_devel/vmlinux vmlinux
gzip -vf9 vmlinux
vmlinux:         53.9% -- replaced with vmlinux.gz
make[2]: Leaving directory `/home/fpe/linuxppc_2_4_devel/arch/ppc/boot/images'
/bin/sh /home/fpe/linuxppc_2_4_devel/scripts/mkuboot.sh -A ppc -O linux -T kernel \
-C gzip -a 00000000 -e 00000000 \
-n 'Linux-2.4.25' \
-d images/vmlinux.gz images/vmlinux.UBoot
Image Name:   Linux-2.4.25
Created:      Mon May  9 13:23:04 2005
Image Type:   PowerPC Linux Kernel Image (gzip compressed)
Data Size:    977813 Bytes = 954.90 kB = 0.93 MB
Load Address: 0x00000000
Entry Point:  0x00000000
ln -sf vmlinux.UBoot images/uImage
rm -f ./mkuboot
make[1]: Leaving directory `/home/fpe/linuxppc_2_4_devel/arch/ppc/boot'
[fpe@s020403 linuxppc_2_4_devel]$
```

The resulting kernel image is: **`./arch/ppc/boot/images/uImage`**. This image could now be used to boot with U-Boot (see previous chapters).

5.4.2.1 Building Linux for the TB5200

For the TB5200 please use a predefined configuration file with “tb5200” in the name (instead of “tqm5200). For example: **`.config_tb5200_2006-06-13_1330`**.

LIN5200.SWM.105.doc

5.4.2.2 Building Linux for the TQM5200S

To build a Linux Kernel for the TQM5200S you could use a predefined configuration file with "tqm5200s" in the name. For example: **`.config_tqm5200s_2006-07-12_1210`**.

5.4.3 Configuring the Linux Kernel

Instead of using an existing configuration file (.config), you could change all kernel parameters manually. Thus you could adapt the kernel exactly to your needs.

As a good starting point you can use the default configuration for the TQM5200 module: "tqm5200_config".

Hint: In the default configuration the voyager frame-buffer graphic driver is selected. On TQM5200 modules without graphic controller the graphic driver must be deselected manually. Otherwise the kernel would not boot properly:

```
# make mrproper
…
# make TQM5200_config
…
# make menuconfig
```

This starts a graphical configuration tool. Now you can modify the configuration according to your needs. When exiting the menu, answer the question "Do you wish to save your new kernel configuration?" with "Yes". Then build the kernel:

```
# make dep
# make uImage
```

The resulting kernel image is **`./arch/ppc/boot/images/uImage`**. This image could now be used to boot with U-Boot (see previous chapters).

If you want to repeat the configuration step, run "make menuconfig" and "make uImage". So only the changes are compiled newly. Depending on the changed configuration it may be required to run "make dep".

5.4.3.1 Available default configurations

The following default configurations are available:

- **`TQM5200_config`** → for the STK52xx baseboard
- **`TB5200_config`** → for the TB5200 baseboard
- **`TQM5200S_config`** → **`for the TQM5200S module`**

5.4.4 TQM5200 specific Kernel Configuration Options

In this chapter mostly the TQM5200 specific Linux configuration options are described. For other "standard" kernel options see the general Linux documentation (for instance at [2] or [7]). The configuration options are shown in "make menuconfig" syntax.

5.4.4.1 SM501 (“voyager”) Frame-buffer Driver

```
Console drivers --->
   Frame-buffer support --->
      [*]   Support for frame buffer devices [EXPERIMENTAL]
      [*]   Silicon Motion VOYAGER support (on TQM5200)
      [*]   Advanced low level driver options
      <*>     8 bpp packed pixels support
      <*>     16 bpp packed pixels support
      <*>     32 bpp packed pixels support
      [*]   Select compiled-in fonts
      [*]     VGA 8x8 font
      [*]     VGA 8x16 font
```

Hint: If this option is selected, the kernel does not boot properly on systems where no SM501 graphic controller is present

5.4.4.2 Graphic CONSOLE over VGA and Keyboard

```
Character devices --->
   [*] Virtual terminal
   [*]   Support for console on virtual terminal
```

Hint: Depending on kernel parameters there may be no CONSOLE via serial interface, if you select this option. If you want console output on both interfaces (VGA and serial line) multiple “console” parameters must be passed to the kernel (over the **`bootargs`** environment variable, see the DULG FAQ section [1]).

5.4.4.3 Activation of the mouse driver (PS/2):

```
Character devices --->
   Mice --->
      <*> Mouse Support (not serial and bus mice)
      [*]   PS/2 mouse (aka "auxiliary device") support
```

Hint: This only works on the STK52xx baseboard, or a derived design (you need the special PS/2 controller chip)

5.4.4.4 SRAM driver:

```
Character devices --->
   <*> Generic SRAM driver
```

Hint: The driver only works on TQM5200 modules with on-board SRAM.

5.4.4.5 Support for USB sticks (Mass Storage Devices)

```
SCSI support --->
   <*> SCSI support
   <*>    SCSI Disk support
USB support --->
   <*> USB Mass Storage support
```

Hint: on the STK52xx only the lower USB port can be used. The upper USB receptacle of X8 is not connected electrically. On the TB5200 both USB ports are available.

5.4.4.6 Baseboard configuration

```
Platform support  --->
   (STK52XX) TQM5200 baseboard in use
      ( ) TB5200
      (*) STK52XX
```

Actually you could choose between two baseboards for the TQM5200:

- STK52xx (default)
- TB5200

Further baseboards may be added in the future. Depending on the baseboard setting, other configuration options are available or hidden.

5.4.4.7 I2C

```
I2C support  --->
   <*> I2C support
   <*> MPC5xxx I2C Algorithm
   <*> MPC5xxx TQM5200 I2C Adapter
   <*> I2C device interface
```

This make it possible for user-space applications to use the I2C bus. For example to access the onboard EEPROM on the TQM5200.

Hint: I2C support has to be activated, if the external (I2C) RTC should be used.

5.4.4.8 External RTC (on I2C bus)

```
Character devices  --->
   <*> Enhanced Real Time Clock Support
   [*] M41T00 Real Time Clock Support
```

Hint: For use of the external RTC, I2C support has to be configured.

With this configuration, the external RTC on the I2C bus is used instead of the build-in RTC of the MPC5200. The external RTC has the advantage that it could be buffered with an battery during power off (with the build-in RTC this is not possible).

The following baseboards have an external I2C RTC:

- STK52xx (Rev 200 and greater)
- TB5200

5.4.4.9 Watchdog driver

```
Character devices  --->
   Watchdog Cards --->
      [*] Watchdog Timer Support
      <*> MPC5XXX Watchdog Timer
```

For a description of the watchdog driver see the man page **`wdt_mpc5xxx.4`** (in the /Documentation/man4/ directory of the kernel source tree).

5.4.4.10 GPIO driver (Status LED, PWM)

```
Character devices  --->
   <*> TQM5200 GPIO driver
```

Currently the GPIO driver supports the following functionality:

- PWM (three PWM channels on the Pins GPIO29, GPIO30 and GPIO31)
- Status-LED (only on the TB5200 baseboard)

For a description of the GPIO driver see the man page **tqm5200_digio.4** (in the /Documentation/man4/ directory of the kernel source tree).

5.4.4.11 German keyboard layout

```
Character devices  --->
   [*] Support for de-latin1 keymap
```

This configures a German keyboard layout for virtual consoles (not for serial consoles!).

5.4.4.12 LM75 temperature sensor (TQM5200S only)

```
Character devices  --->
   I2C support  --->
      <*> I2C support
      <*> MPC5xxx I2C Algorithm
      <*> MPC5xxx TQM5200 I2C Adapter
      I2C Chips support  --->
.........[*] Enable chips support
.........<*>   Support for lm75 chip
      <*> I2C /proc interface (required for hardware sensors)
```

The current temperature could then be requested via the /proc interface:

```
bash-2.05b# cat /proc/sys/dev/sensors/lm75-i2c-0-48/temp
80.0 75.0 38.0
bash-2.05b#
```

The last number in the output line is the current temperature (38.0°C).

5.4.4.13 New Flash-Map (for TQM5200S and Rev B Modules)

All Rev B modules (with MPC5200B CPU) also have a new Flash type soldered. The new Flashes are ‘N’-Type Flashes from Spansion (with a doubled sector size compared to the since then used ‘M’-Typ Flashes).

Because of the doubled sector size, a new Flash-Map is required for this modules (see also chapter 8.2.2 Flash Memory Map).The new Flash-Map could be configured with the following option:

```
Memory Technology Devices (MTD)  --->
   Mapping drivers for chip access  --->
      <*>   CFI Flash device mapped on TQM5200
      [*]     Flash map for Rev B modules
```

5.5 Building U-Boot

If you want to adjust U-Boot to your demands, for instance to reduce memory footprint, to decrease boot time or to enable a certain feature, you had to compile your own U-Boot image. With the ELDK it is very easy to build U-Boot from the sources.

5.5.1 Installing the Sources

Before you could build U-Boot, you have to install the sources on your Linux host. A good place to do this is your home directory (you don’t need root rights to install and build U-Boot).

5.5.1.1 Installing from a tar Archive

The easiest way to install the Linux sources on your host is to use a tar archive. For example **u-boot_2005-04-29_235959.tar.gz** from the TQ-CD:

LIN5200.SWM.105.doc

```
# cd <your homedir>
# cp <path on cd>/u-boot_2005-04-29_235959.tar.gz ./
# tar -xzvf u-boot_2005-04-29_235959.tar.gz
…
```

Update: You could also use the more up-to-date archive **u-boot_tqm5200_2005-11-04_000000.tar.gz** which could be found on the TQ-CD, too.

5.5.1.2 Installing from the CVS Server

Getting the sources from the public CVS-Server is not supported any longer. Use the git archive instead.

5.5.1.2.1 Installing from git archive

“git” is a “directory content manager” used and developed by Linus Torvalds and the Linux Community for the Linux kernel. U-Boot now also uses git.

You can browse the DENX "git" repositories with a web browser (see [10]).

Or you could clone the git archive to your local host. This requires that “git” and “cogito” is installed on your host.

Clone the U-Boot git archive via the git protocol:

```
#cg-clone git://source.denx.net/git/u-boot.git <installdir>
…
```

Hint: If you are sitting behind a firewall, make sure that the "git" port (port 9418) is not blocked.

Clone the U-Boot git archive via http:

```
#cg-clone http://source.denx.net/git/u-boot.git <installdir>
```

5.5.1.3 Where could I get recent sources?

The source tar archives on the TQ-CD are a good starting point for playing with U-Boot on the TQM5200, but they are probably not the most recent sources available.

For current sources please use the DENX git archive (as described above).

Because it may take some time till changes made it into the DENX git archive, please contact TQ-Components directly for the latest sources or patches available (website: http://www.tqc.de, or via email or telephone).

```
…
```

5.5.1.4 Patching the Sources

Patching the sources is only necessary, if you use the source archive **u-boot_2005-04-29_235959.tar.gz**. The updated archive **u-boot_tqm5200_2005-11-04_000000.tar.gz** already contains the patches, so they don’t have to be applied.

The patches could be found on the TQ-CD. To apply a patch the **patch** command is used.

```
host# cd <your homedir>/u-boot
host# patch -p0 < <path on cd>/patch_u-boot_tqm5200_2005-04-07_rtc_mkr
patching file include/configs/TQM5200.h
host# patch -p0 < <path on cd>/patch_u-boot_tqm5200_2005-04-08_sm501_detect_mkr
patching file board/tqm5200/tqm5200.c
host# patch -p0 < <path on cd>/patch_u-boot_tqm5200_2005-04-28_fkt_mkr
patching file board/tqm5200/Makefile
patching file include/mpc5xxx.h
patching file include/configs/TQM5200.h
```

Also the source file **cmd_stk52xx.c** (contained in an tar archive) has to be copied to the U-Boot source directory (needed by the patch **patch_u-boot_tqm5200_2005-04-28_fkt_mkr**).

```
host# tar -xzvf <path on cd>/cmd_stk52xx.c.tar.gz
board/tqm5200/cmd_stk52xx.c
```

5.5.2 Building U-Boot with a default Configuration

Make sure, that ELDK is installed and the environment variables $PATH and $CROSS_COMPILE are set up properly. $PATH must contain the path where you installed ELDK.

```
host# echo $PATH
/opt/eldk/usr/bin:/opt/eldk/bin:/sbin:/usr/sbin:/bin:/usr/bin:/usr/X11R6/bin:/usr/loc
al/bin
host# echo $CROSS_COMPILE
ppx_6xx-
```

Change into the root directory of the U-Boot source tree and do a complete cleanup of the sources (just to be sure):

```
host# cd <your homedir>/u-boot
host# make mrproper
```

Now we could configure U-Boot for the suitable target hardware. In chapter “Specification” you could find a list with all possible configurations for the TQM5200 module. We choose the “automatic” configuration, which should run with all module variants.

```
host# make TQM5200_auto_config
... with automatic CS configuration
Configuring for TQM5200 board...
```

Hint: Currently the default configuration is designed to run your TQM5200 module on a STK52xx.100 baseboard. If you want to use a STK52xx.200 or another baseboard (e. g. TB5200) you have to change the configuration manually (see the chapter “Configuring U-Boot).

After that we can start the compilation. The dependencies are calculated automatically every time the build process starts. According to your system performance the compilation could take several seconds to a few minutes.

```
host# make
…
```

The resulting binary U-Boot image is **./u-boot.bin**. Also an image in s-record format is generated: **./u-boot.srec**.

5.5.2.1 Building U-Boot for the TB5200 baseboard

The TB5200 baseboard is supported by an own configuration file: ./include/configs/TB5200.h.

LIN5200.SWM.105.doc

```
host# make mrproper
…
host# make TB5200_config
... with automatic CS configuration
Configuring for TB5200 board...
host# make
```

Hint: For the TB5200 Rev. 300 and above use `make TB5200_B_config` for configuration (because inside the Tinybox a TQM5200 Rev. B module is used)

5.5.2.2 Building U-Boot for Rev B modules

There is a default configuration for the new Rev B modules (with MPC5200B CPU and new 'N'-Type flashes)

```
host# make mrproper
…
host# make TQM5200_B_config
... with MPC5200B processor
... with automatic CS configuration
Configuring for TQM5200 board...
```

5.5.2.3 Building U-Boot for a TQM5200S

The TQM5200S module is supported by the configuration for the new Rev B Modules.

5.5.3 Configuring U-Boot

Instead of using a predefined configuration, U-Boot could be configured manually. With the numerous configuration options it is possible to adapt U-Boot exactly to your needs.

The configuration file for the TQM5200 module is **`./include/configs/TQM5200.h`**. There you could do the configuration changes. A list of possible options can be found in [4]. After you have changed the configuration, U-Boot must be re-build:

```
host# make
…
```

In the following subchapters you could find some configuration examples.

5.5.3.1 Configuring U-Boot for a STK52xx.200 baseboard

The current default configuration is intended to run on a STK52xx.100 baseboard. If you would like to run your TQM5200 module on a STK52xx.200 baseboard (to use the added hardware features), you have to change the configuration.

To do that change the following line in **`./include/configs/TQM5200.h`** (the changes are shown in unified diff format):

```
-#define CONFIG_STK52XX_REV100   1      /*  define for revision 100 baseboards */
+#undef CONFIG_STK52XX_REV100           /*  define for revision 100 baseboards */
```

After that you have to re-build U-Boot:

```
host# make
…
```

The resulting U-Boot now supports the added hardware features of revision 200 of the STK52xx (for instance you could play .WAV files).

5.5.3.2 Building a small and fast U-Boot

The default U-Boot is configured to support as much of the TQM5200 (and STK52xx) hardware as possible. This is intended to show all the features of the hardware. Therefore this U-Boot configuration is not optimized for size or speed.

If you don't need to support all hardware interfaces in U-Boot (for instance, because U-Boot is just used to boot a Linux system which then supports the hardware) you could remove the configuration for those parts and get a smaller and faster U-Boot.

A sample U-Boot configuration for a minimum setup (without any hardware support except for serial line, flash and Ethernet) could be found in the file **`TQM5200_2005-05-10_minimal.h`** on the TQ-CD.

Replace the existing configuration file with this new file and re-build U-Boot

```
host# cd <your homedir>/u-boot
host# mv include/configs/TQM5200.h include/configs/TQM5200.h.orig
host# cp <path on cd>/TQM5200_2005-05-10_minimal.h ./include/configs/TQM5200.h
host# make
…
```

The resulting U-Boot image size is only about 128 kByte (compared to nearly 300 kByte for the full configuration) and boot time is minimized (the longest part in boot time is now waiting for the Ethernet autonegotiation process to finish).

5.5.4 Using a backup copy of U-Boot in flash (high-boot option)

Normally the TQM5200 is configured to do a low-boot. This means U-Boot is located and started at the beginning of the flash address space. On newer Versions of the TQM5200 also a high-boot option is available. If high-boot is selected, an U-Boot image at the end of the flash address space is started.

The intention of the high-boot option is to provide a fall back boot option during the evaluation and development phase. Especially for people without access to a BDI2000 debugger or another possibility to reprogram the flash via JTAG/BDM. The normal low-boot U-Boot and the high-boot U-Boot could be held in parallel in Flash. If something goes wrong with the low-boot U-Boot (e. g if a newly build U-Boot version doesn't boot any longer), the system could be booted by the high-boot U-Boot. And since the system is running again, the low-boot U-Boot could be updated with a known to be running copy.

Hardware supporting the high-boot option:

- TQM5200S modules
- TQM5200 "Rev B" modules (Rev 214 and above)

Switching between low-boot and high-boot

Over a pin of the TQM5200 the desired boot option (low- or high-boot) could be selected. Please refer to the TQM5200 hardware manual for a detailed description.

On the STK52xx the desired boot option could be selected with the jumper X30.

- Jumper X30 is open → low-boot
- Jumper X30 is closed → high-boot

Building a high-boot U-Boot

To build a high-boot Version of U-Boot, use the following default configuration:

```
host# make TQM5200_B_HIGHBOOT_config
... with MPC5200B processor
... with automatic CS configuration
Configuring for TQM5200 board...
host# make
...
```

A precompiled high-boot U-Boot Image could be found on the TQ-CD:
u-boot_tqm5200_revb_2006-07-06_highboot_mkr.bin

U-Boot Environment

Both U-Boot versions (low-boot and high-boot) are using the same environment, stored in flash at address 0xFC080000.

Storing a high-boot U-Boot to flash

U-Boot must be stored at the start of the last Megabyte of flash memory. The exact address depends on the total amount of flash soldered to the TQM5200.

Flash size	Flash start address	Flash end address	Start address of high-boot U-Boot
4 Mbyte	0xFC000000	0xFC3FFFFF	0xFC300000
16 Mbyte	0xFC000000	0xFCFFFFFF	0xFCF00000
32 Mbyte	0xFC000000	0xFDFFFFFF	0xFDF00000

Example for a TQM5200S-AA (with 32 Mbyte flash):

```
=> flinfo

Bank # 1: CFI conformant FLASH (32 x 16)  Size: 32 MB in 128 Sectors
 Erase timeout 16384 ms, write timeout 2 ms, buffer write timeout 5 ms, buffer s
ize 32
  Sector Start Addresses:
 FC000000  RO  FC040000  RO  FC080000  RO  FC0C0000 E    FC100000 E
 FC140000 E    FC180000 E    FC1C0000 E    FC200000 E    FC240000 E
 FC280000 E    FC2C0000 E    FC300000 E    FC340000 E    FC380000 E
 FC3C0000 E    FC400000 E    FC440000 E    FC480000 E    FC4C0000 E
 FC500000 E    FC540000 E    FC580000 E    FC5C0000 E    FC600000 E
 FC640000 E    FC680000 E    FC6C0000 E    FC700000 E    FC740000 E
 FC780000 E    FC7C0000 E    FC800000 E    FC840000 E    FC880000 E
 FC8C0000 E    FC900000 E    FC940000 E    FC980000 E    FC9C0000 E
 FCA00000 E    FCA40000 E    FCA80000 E    FCAC0000 E    FCB00000 E
 FCB40000 E    FCB80000 E    FCBC0000 E    FCC00000 E    FCC40000 E
 FCC80000 E    FCCC0000 E    FCD00000 E    FCD40000 E    FCD80000 E
 FCDC0000 E    FCE00000 E    FCE40000 E    FCE80000 E    FCEC0000 E
 FCF00000 E    FCF40000 E    FCF80000 E    FCFC0000 E    FD000000 E
 FD040000 E    FD080000 E    FD0C0000 E    FD100000 E    FD140000 E
 FD180000 E    FD1C0000 E    FD200000 E    FD240000 E    FD280000 E
```

```
 FD2C0000 E    FD300000 E    FD340000 E    FD380000 E    FD3C0000 E
 FD400000 E    FD440000 E    FD480000 E    FD4C0000 E    FD500000 E
 FD540000 E    FD580000 E    FD5C0000 E    FD600000 E    FD640000 E
 FD680000 E    FD6C0000 E    FD700000 E    FD740000 E    FD780000 E
 FD7C0000 E    FD800000 E    FD840000 E    FD880000 E    FD8C0000 E
 FD900000 E    FD940000 E    FD980000 E    FD9C0000 E    FDA00000 E
 FDA40000 E    FDA80000 E    FDAC0000 E    FDB00000 E    FDB40000 E
 FDB80000 E    FDBC0000 E    FDC00000 E    FDC40000 E    FDC80000 E
 FDCC0000 E    FDD00000 E    FDD40000 E    FDD80000 E    FDDC0000 E
 FDE00000 E    FDE40000 E    FDE80000 E    FDEC0000 E    FDF00000
 FDF40000      FDF80000 E    FDFC0000 E
=> erase 0xFDF00000 0xFDF7FFFF

.. done
Erased 2 sectors
=> tftp 200000 tqm5200/u-boot_tqm5200_revb_2006-07-06_highboot_mkr.bin
Using FEC ETHERNET device
TFTP from server 172.20.1.31; our IP address is 172.20.5.71
Filename 'tqm5200/u-boot_tqm5200_revb_2006-07-06_highboot_mkr.bin'.
Load address: 0x200000
Loading: #################################################################
         ##
done
Bytes transferred = 341404 (5359c hex)
=> cp.b 0x200000 0xFDF00000 ${filesize}
Copy to Flash... done
=>
```

5.6 Updating U-Boot

After building a new U-Boot image you would like to download it to your hardware. In this chapter the process of updating U-Boot over Ethernet is described.

Hint: Because you have to delete the current U-Boot from the flash of your target, before you can update the new U-Boot, there is a risk to make your system unusable, if something in the update process goes wrong! If U-Boot doesn't boot any more, your only chance to make your hardware run again is with a JTAG flash programming tool (for instance a BDI2000 debugger).

To make the update process more comfortable some environment variables are predefined:

```
=> print load
load=tftp 200000 $(u-boot)
=> print update
      update=protect off FC000000 FC05FFFF;erase FC000000 FC05FFFF;cp.b 200000
FC000000 $(filesize);protect on FC000000 FC05FFFF
=>
```

Copy the U-Boot image which should be updated to the root directory of your TFTP server. Then set the name of the new U-Boot image:

```
=> set u-boot u-boot_tqm5200_2005-05-10_minimal.bin
```

After that you can download the new file to the target RAM:

```
=> run load
Using FEC ETHERNET device
TFTP from server 172.20.5.13; our IP address is 172.20.5.73
Filename 'u-boot_tqm5200_2005-05-10_minimal.bin'.
Load address: 0x200000
Loading: ##########################
done
Bytes transferred = 132084 (203f4 hex)
```

Don’t proceed with the next step if something went wrong during this step!

The next step deletes the old U-Boot from flash and copies the new image to flash.

```
=> run update
Un-Protected 6 sectors

...... done
Erased 6 sectors
Copy to Flash... done
Protected 6 sectors
```

Hint: Before Updating U-Boot, be sure that the new U-Boot image doesn’t exceed the reserved area for U-Boot in flash memory (see chapter specification for the flash map).

After a reset the updated U-Boot should boot.

5.7 Adapting U-Boot to a new Baseboard

If you have created your own baseboard for the TQM5200 module you need a new U-Boot configuration to support your new hardware.

Please always use a new configuration file if you use another hardware than STK5200.

If you want to support completely new hardware which is not supported in U-Boot, yet, you could write your own drivers and integrate it in U-Boot. If you do that, don’t forget to send patches with your changes back to the U-Boot community (exactly to the U-Boot users list: u-boot-users@lists.sourceforge.net). This has the advantage, that your changes could be integrated into the U-Boot main source tree – and in new U-Boot versions your enhancements are then automatically included.

6. Working with U-Boot

This chapter describes the run-time configuration of U-Boot and the use of some U-Boot commands on the STK52xx.

6.1 Console Input and Output

In the default condition, the U-Boot is configured so, that input and output is handled via the serial interface.

The configuration for the input and output device can be changed at run-time:

`=> set stdin kbd`	->	sets the input to the PS/2 keyboard
`=> set stdin serial`	->	sets the input to the serial interface (PSC1)
`=> set stdout vga`	->	sets the output to VGA (CRT and LCD)
`=> set stdout serial`	->	sets the output to the serial interface (PSC1)
`=> set stdout serial1`	->	sets the output to the second serial interface (PSCx) Which PSC is used for the second serial interface depends on the used base board (PSC2 on STK52xx and PSC6 on TB5200)

Hint: VGA could be selected only if your TQM5200 module has a graphic chip (some variants don't have).

Hint: The PS/2 keyboard could be selected only if a keyboard was detected during boot time.

Hint: The support for multiple serial interfaces (**`serial0`** and **`serial1)`** is only supported by newer U-Boot versions (see /doc/README.serial_multi in the U-Boot source tree).

These settings are only effective, if the environment variables were already existent during boot time. Thus after setting the variables the first time you must reset the board to make your changes effective (don't forget the **`saveenv`** command). After that you could change the current input or output device at run-time.

To make your changes permanent, you have to save the environment variables with the **`saveenv`** command.

6.2 The I2C Subsystem

6.2.1 Probing for I2C devices

The **`iprobe`** command could be used to scan the I2C bus for connected devices. Internally the command tries to send an I2C packet to all possible I2C bus addresses. Every device address for which an acknowledge is received is printed on the console.

```
=> iprobe
Valid chip addresses: 50 68 7F
```

6.2.2 On-board EEPROM

The on-board EEPROM is located at I2C bus address 50. There are two command classes to access the EEPROM:

LIN5200.SWM.105.doc

- **eeprom**
- **imd** and **imw**

With the **eeprom** command you have direct access to the on-board EEPROM similar to the **cp** command with flash memory. The **eeprom** command copies data from the EEPROM to RAM or vice versa:

```
=> md 200000
00200000: 27051956 9b52c2bf 4253d79b 0000060b    '..V.R..BS......
00200010: 00000000 00000000 60c2f6a1 05070600    ........`.......
00200020: 54514d35 3230303a 20736574 20646566    TQM5200: set def
00200030: 61756c74 20656e76 69726f6e 6d656e74    ault environment
00200040: 00000603 00000000 6563686f 20277365    ........echo 'se
00200050: 7420656e 7669726f 6e6d656e 7420666f    t environment fo
00200060: 7220626f 6f74696e 67206f76 6572204e    r booting over N
00200070: 4653270a 73657465 6e762066 6c617368    FS'.setenv flash
00200080: 5f6e6673 20277275 6e206e66 73617267    _nfs 'run nfsarg
00200090: 73206164 64697020 61646463 6f6e7320    s addip addcons
002000a0: 6164646d 6973633b 20626f6f 746d2024    addmisc; bootm $
002000b0: 286b6572 6e656c5f 61646472 29270a73    (kernel_addr)'.s
002000c0: 6574656e 76206e65 745f6e66 73202774    etenv net_nfs 't
002000d0: 66747020 24286c6f 61646164 64722920    ftp $(loadaddr)
002000e0: 2428626f 6f746669 6c65293b 2072756e    $(bootfile); run
002000f0: 206e6673 61726773 20616464 69702061     nfsargs addip a
=> eeprom write 200000 0 100

EEPROM @0x50 write: addr 00200000  off 0000  count 256 ... done
=> eeprom read 300000 0 100

EEPROM @0x50 read: addr 00300000  off 0000  count 256 ... done
=> md 300000
00300000: 27051956 9b52c2bf 4253d79b 0000060b    '..V.R..BS......
00300010: 00000000 00000000 60c2f6a1 05070600    ........`.......
00300020: 54514d35 3230303a 20736574 20646566    TQM5200: set def
00300030: 61756c74 20656e76 69726f6e 6d656e74    ault environment
00300040: 00000603 00000000 6563686f 20277365    ........echo 'se
00300050: 7420656e 7669726f 6e6d656e 7420666f    t environment fo
00300060: 7220626f 6f74696e 67206f76 6572204e    r booting over N
00300070: 4653270a 73657465 6e762066 6c617368    FS'.setenv flash
00300080: 5f6e6673 20277275 6e206e66 73617267    _nfs 'run nfsarg
00300090: 73206164 64697020 61646463 6f6e7320    s addip addcons
003000a0: 6164646d 6973633b 20626f6f 746d2024    addmisc; bootm $
003000b0: 286b6572 6e656c5f 61646472 29270a73    (kernel_addr)'.s
003000c0: 6574656e 76206e65 745f6e66 73202774    etenv net_nfs 't
003000d0: 66747020 24286c6f 61646164 64722920    ftp $(loadaddr)
003000e0: 2428626f 6f746669 6c65293b 2072756e    $(bootfile); run
003000f0: 206e6673 61726773 20616464 69702061     nfsargs addip a
=>
```

The **imd** and **imw** commands offer a more device independent way of accessing the eeprom devices. Thus some more parameters must be specified to address the device:

```
=> help imd
imd chip address[.0, .1, .2] [# of objects]
    - i2c memory display

=> help imw
imw chip address[.0, .1, .2] value [count]
    - memory write (fill)
```

In the following example 32 (0x20) bytes are read from the I2C device with address 50 (EEPROM) beginning at offset 0. The I2C command is composed from 2 address bytes.

```
=> imd 50 0.2 20
0000: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
```

Then 5 bytes are written to the same device starting at offset 7. The I2C command is composed from 2 address bytes.

```
=> imw 50 7.2 5a 5
=> imd 50 0.2 20
0000: 00 00 00 00 00 00 00 5a 5a 5a 5a 5a 00 00 00 00    .......ZZZZZ....
0010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
```

6.3 Sound Commands

The sound commands only work on a STK52xx revision 200 baseboard. Revision 100 of the STK52xx lacks the sound hardware. To be able to use the sound commands you have to ensure that U-Boot is configured correctly (option **CONFIG_STK52XX_REV100** must not be set and option **CFG_CMD_BSP** has to be set).

6.3.1 Interface

Connector X59 is used for the audio output. Two speakers with 4-8 Ohm impedance could be connected directly.

6.3.2 Setting the volume

The volume is set consistently for all sound commands by setting the environment variable **volume**. If **volume** is not set a default volume of 45 is used.

```
=> setenv volume 50
```

This sets the volume to 50. Note that the variable is only changed in RAM. If you want to make your changes permanent, you have to do **saveenv.**

volume can be set between 0 and 63. 63 is the loudest.

6.3.3 beep – play a short Beep

The **beep** command plays a short beep on the specified channel.

```
=> help beep
beep [channel]
    - play short beep on "l"eft or "r"ight channel
```

If no channel is specified, the beep is played on both channels.

```
=> beep l
Beep on left channel
=> beep
Beep on left and right channel
```

Hint: Because the I2S controller sometimes lost synchronization the beep could be issued on the wrong channel.

6.3.4 wav – play a wav File

The **wav** command plays Microsoft ".wav" files. Currently only the format 16 Bit, 44200 kHz is supported.

LIN5200.SWM.105.doc

```
=> help wav
wav [addr] [bytes]
    - play wav file at address 'addr' with length 'bytes'
```

If the parameters **addr** and **bytes** are omitted, the values are taken from the environment variables **loadaddr** and **filsize**.

```
=> tftp 200000 synth.wav
Using FEC ETHERNET device
TFTP from server 172.20.5.13; our IP address is 172.20.5.73
Filename 'synth.wav'.
Load address: 0x200000
Loading: ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ####################
done
Bytes transferred = 3432762 (34613a hex)
=> wav 200000 34613a
Play wave file at 0x200000 with length 0x34613a
```

6.3.5 sound – generate Sounds

The **sound** command generates different sounds. It is also possible to access the registers of the DA converter (PCM1772) directly.

```
=> help sound
sound saw [duration] [freq] [channel]
    - generate sawtooth for 'duration' ms with frequency 'freq'
      on left "l" or right "r" channel
sound square [duration] [freq] [channel]
    - generate squarewave for 'duration' ms with frequency 'freq'
      on left "l" or right "r" channel
pcm1772 reg val
```

Hint: Because the I2S controller sometimes lost synchronization the sound could be issued on the wrong channel.

6.4 Setting the LEDs

6.4.1 STK52xx

The led command only works on a STK52xx revision 200 baseboard. To be able to use the led command you have to ensure that U-Boot is configured correctly (option **CONFIG_STK52XX_REV100** must not be set and option **CFG_CMD_BSP** has to be set).

With the **fkt led** command the LEDs on the STK52xx.200 could be switched on or off.

```
=> help fkt
fkt led number on/off
     - 'number's like printed on SKT52XX board
```

The numbers of the LEDs are according to the printed labels on the STK52xx board.

```
=> fkt led 48 on
=> fkt led 4 off
```

Hints:

- Jumper X15 („LED-EN“) has to be set, to power the LEDs
- After switching on the power supply it could happen that some LEDs are already glowing. This is because the LED ports are not configured until the **`fkt led`** command is issued the first time. Before that the pins have their reset state.

6.4.2 TB5200

With the **`led`** command the status LEDs on the TB5200 could be switched on or off.

```
=> help led
led on/off
```

6.5 Network Subsystem

U-Boot supports a bunch of networking commands. To keep things simple, no complete TCP/IP stack is implemented.

6.5.1 tftp – trivial file transmit protocol to download images to the target

The **`tftp`** command allows you to download an image from your tftp-server to the target.

```
=> tftp 200000 uImage_tqm5200_2005-05-04_basic
Using FEC ETHERNET device
TFTP from server 172.20.5.13; our IP address is 172.20.5.73
Filename 'uImage_tqm5200_2005-05-04_basic'.
Load address: 0x200000
Loading: #################################################################
         #################################################################
         ##############################################################
done
Bytes transferred = 977879 (eebd7 hex)
=>
```

6.5.2 ping – send ICMP ECHO_REQUEST to Network Host

The **`ping`** command sends ICMP ECHO_REQUEST to a Network Host. If the host answers the request an alive message is printed.

```
=> ping 172.20.5.13
Using FEC ETHERNET device
host 172.20.5.13 is alive
```

Note that U-Boot does not answer to ICMP ECHO_REQUESTs. So if you ping to a U-Boot target you will get no answer.

6.5.3 nfs – load Images via Network using NFS Protocol

Instead of loading Images from a TFTP server, the NFS protocol could be used to load images from a NFS server.

```
=> help nfs
nfs [loadAddress] [host ip addr:bootfilename]

=> nfs 200000 172.20.5.13:/opt/eldk/ppx_6xx/images/uRamdisk
```


LIN5200.SWM.105.doc

```
Using FEC ETHERNET device
File transfer via NFS from server 172.20.5.13; our IP address is 172.20.5.73
Filename '/opt/eldk/ppx_6xx/images/uRamdisk'.
Load address: 0x200000
Loading: #################################################################
         #################################################################
         #################################################################
         #################################################################
         ##################################################
done
Bytes transferred = 1581115 (18203b hex)
=> imi 200000

## Checking Image at 00200000 ...
   Image Name:   Simple Embedded Linux Framework
   Created:      2004-11-10   9:45:02 UTC
   Image Type:   PowerPC Linux RAMDisk Image (gzip compressed)
   Data Size:    1581051 Bytes =  1.5 MB
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK
```

6.5.4 dhcp - invoke DHCP client to obtain IP/boot params

With the **dhcp** command it is possible to automatically get IP and boot parameters from a DHCP server.

If the environment variable **autoload** is set to a value starting with “n”, the **dhcp** command will just perform a lookup of the configuration from the DHCP server, but not try to load any image using TFTP.

```
=> dhcp
BOOTP broadcast 1
DHCP client bound to address 172.20.10.51
```

Hint: If **dhcp** is invoked and there is no response from a DHCP server in the network, U-Boot endlessly repeats the requests.

See [4] for further information.

6.6 Real Time Clock

With the **date** command the RTC could be set.

The input format for the date command is like the format of the Linux date command:

date [MMDDhhmm[[CC]YY][.ss]]

```
=> date 051209112005.50
Date: 2005-05-12 (Thursday)    Time:  9:11:50
=> date
Date: 2005-05-12 (Thursday)    Time:  9:11:54
```

The internal RTC of the MPC5200 doesn't have a separate pin to power the RTC. If the main power is switched off, time gets lost. As a workaround on newer TQM5200 baseboards an external RTC is added, connected via the I2C bus. This RTC is buffered by the on-board battery, so time is preserved when the power supply is switched of.

The following baseboards have an external RTC on it:

- STK52xx (Rev. 200 and greater)
- TB5200

To use the external RTC the following configuration options have to be set in the U-Boot configuration file.

```
# define CONFIG_RTC_M41T11       1
# define CFG_I2C_RTC_ADDR        0x68
# define CFG_M41T11_BASE_YEAR    1900
```

and

```
# undef CONFIG_RTC_MPC5200
```

6.6.1 Synchronizing the RTC via Network

If there is a running ntp (network time protocol) server in your network, you could use this server to synchronize the RTC:

```
=> date
Date: 2005-04-20 (Wednesday)    Time: 10:35:12
=> sntp 172.20.2.11
Date: 2005-05-18 Time: 14:52:57
=> date
Date: 2005-05-18 (Wednesday)    Time: 14:52:59
```

6.7 IDE Subsystem

U-Boot supports commands to access devices on the IDE bus.

The commands of the IDE subsystem are started with **ide** followed by a subcommand:

```
=> help ide
ide reset - reset IDE controller
ide info  - show available IDE devices
ide device [dev] - show or set current device
ide part [dev] - print partition table of one or all IDE devices
ide read  addr blk# cnt
ide write addr blk# cnt - read/write `cnt' blocks starting at block `blk#'
    to/from memory address `addr'
```

During power-up U-Boot already prints information about the detected IDE devices:

```
…
POST cpu PASSED
IDE:   Bus 0: OK
  Device 0: Model: TOSHIBA THNCF128MBA Firm: 2.20 Ser#: STCB21M83007C12555C1
            Type: Removable Hard Disk
            Capacity: 122.2 MB = 0.1 GB (250368 x 512)
  Device 1: Model: HITACHI_DK238A-43 Firm: 0050A0A0 Ser#: E08691
            Type: Hard Disk
            Capacity: 4126.9 MB = 4.0 GB (8452080 x 512)
SRAM:  1 MB
…
```

The MPC5200 only supports one IDE channel (the primary channel). The master device is named “Device 0”, the slave device “Device 1”.

If you want to use a CompactFlash card on the STK52xx, the Jumper X79 (labeled with “MSTR CF”) has to be set.

If IDE devices (for instance a IDE hard disk or a IDE CD-Rom) are connected to the STK52xx via X48, then the individual Jumpers on the devices determine if the device is behaving as master or as slave.

Hint: Make sure that you don't use two master devices or two slave devices at the same time.

6.7.1 Getting Information about IDE Devices

The commands **`ide info`** and **`ide part`** could be used to show some information about the connected IDE devices:

```
=> ide info

IDE device 0: Model: TOSHIBA THNCF128MBA Firm: 2.20 Ser#: STCB21M83007C12555C1
            Type: Removable Hard Disk
            Capacity: 122.2 MB = 0.1 GB (250368 x 512)
IDE device 1: Model: HITACHI_DK238A-43 Firm: 0050A0A0 Ser#: E08691
            Type: Hard Disk
            Capacity: 4126.9 MB = 4.0 GB (8452080 x 512)
=> ide part

Partition Map for IDE device 0  --   Partition Type: DOS

Partition     Start Sector     Num Sectors     Type
    1                32           31456       4
    2             31488          218880      83


Partition Map for IDE device 1  --   Partition Type: DOS

Partition     Start Sector     Num Sectors     Type
    1                63           48132      83
    2             48195          530145      82
    3            578340         7871850      83
```

6.7.2 Accessing IDE Devices

With the commands **`ide read`** and **`ide write`** whole blocks of data could be read or written to a IDE device. In the following example the MBR (master boot record) of device 1 is copied to address 0x200000 in SDRAM:

```
  => ide device 1

IDE device 1: Model: HITACHI_DK238A-43 Firm: 0050A0A0 Ser#: E08691
            Type: Hard Disk
            Capacity: 4126.9 MB = 4.0 GB (8452080 x 512)
... is now current device
=> ide read 200000 0 1

IDE read: device 1 block # 0, count 1 ... 1 blocks read: OK
=> md 200000
00200000: eb4890d0 bc007cfb 5007501f fcbe1b7c    .H....|.P.P....|
00200010: bf1b0650 57b9e501 f3a4cbbe be07b104    ...PW...........
00200020: 382c7c09 751583c6 10e2f5cd 188b148b    8,|.u...........
00200030: ee83c610 49741638 2c74f6be 10070302    ....It.8,t......
00200040: 80000080 57820000 0008faea 507c0000    ....W.......P|..
00200050: 31c08ed8 8ed0bc00 20fba040 7c3cff74    1.......  ..@|<.t
00200060: 0288c252 be767de8 3401f6c2 807454b4    ...R.v}.4....tT.
00200070: 41bbaa55 cd135a52 724981fb 55aa7543    A..U..ZRrI..U.uC
00200080: a0417c84 c0750583 e1017437 668b4c10    .A|..u....t7f.L.
00200090: be057cc6 44ff0166 8b1e447c c7041000    ..|.D..f..D|....
002000a0: c7440201 0066895c 08c74406 00706631    .D...f.\..D..pf1
```

```
002000b0: c0894404 6689440c b442cd13 7205bb00    ..D.f.D..B..r...
002000c0: 70eb7db4 08cd1373 0af6c280 0f84f300    p.}....s........
002000d0: e98d00be 057cc644 ff006631 c088f040    .....|.D..f1...@
002000e0: 66894404 31d288ca c1e20288 e888f440    f.D.1..........@
002000f0: 89440831 c088d0c0 e8026689 0466a144    .D.1......f..f.D
=>
00200100: 7c6631d2 66f73488 540a6631 d266f774    |f1.f.4.T.f1.f.t
00200110: 0488540b 89440c3b 44087d3c 8a540dc0    ..T..D.;D.}<.T..
00200120: e2068a4c 0afec108 d18a6c0c 5a8a740b    ...L......l.Z.t.
00200130: bb00708e c331dbb8 0102cd13 722a8cc3    ..p..1......r*..
00200140: 8e06487c 601eb900 018edb31 f631fffc    ..H|`......1.1..
00200150: f3a51f61 ff26427c be7c7de8 4000eb0e    ...a.&B|.|}.@...
00200160: be817de8 3800eb06 be8b7de8 3000be90    ..}.8.....}.0...
00200170: 7de82a00 ebfe4752 55422000 47656f6d    }.*...GRUB .Geom
00200180: 00486172 64204469 736b0052 65616400    .Hard Disk.Read.
00200190: 20457272 6f7200bb 0100b40e cd10ac3c     Error.........<
002001a0: 0075f4c3 00000000 00000000 00000000    .u..............
002001b0: 00000000 00000000 00000000 00000001    ................
002001c0: 010083fe 3f023f00 000004bc 00000000    ....?.?.........
002001d0: 010382fe 3f2343bc 0000e116 08000000    ....?#C.........
002001e0: 012483fe bf0d24d3 08006a1d 78000000    .$....$...j.x...
002001f0: 00000000 00000000 00000000 000055aa    ..............U.
```

Because working with the **`ide read`** and **`ide write`** commands is not very comfortable when accessing whole files from a filesystem, U-Boot implements commands to directly access FAT, ext2, VFAT. cramfs. reiser and jffs2 filesystems.

The commands **`fatls`** and **`ext2ls`** could be used to list files in a filesystem (FAT or ext2):

```
=> help fatls
fatls <interface> <dev[:part]> [directory]
    - list files from 'dev' on 'interface' in a 'directory'

=> help ext2ls
ext2ls <interface> <dev[:part]> [directory]
    - list files from 'dev' on 'interface' in a 'directory'
```

In the following example the contents of an ext2 filesystem on IDE device 1 in partition 3 are printed:

```
=> ext2ls ide 1:3
<DIR>       4096 .
<DIR>       4096 ..
<DIR>       4096 lost+found
<DIR>       4096 etc
<DIR>       4096 media
<SYM>         11 cdrom
<DIR>       4096 var
<DIR>       4096 usr
<DIR>       4096 bin
<DIR>       4096 boot
<DIR>      24576 dev
<DIR>       4096 home
<DIR>       4096 lib
<DIR>       4096 mnt
<DIR>       4096 proc
<DIR>       4096 root
<DIR>       8192 sbin
<DIR>       4096 tmp
<DIR>       4096 sys
<DIR>       4096 srv
<DIR>       4096 opt
<DIR>       4096 initrd
```

With the commands **fatload** and **ext2load** files could be loaded from an IDE device to memory:

```
=> help ext2load
ext2load <interface> <dev[:part]> [addr] [filename] [bytes]
    - load binary file 'filename' from 'dev' on 'interface'
      to address 'addr' from ext2 filesystem
```

In the following example the file **/etc/fb.modes** is copied from the ext2 filesystem on IDE device 1, partition 3 to address 0x200000 in SDRAM:

```
=> ext2load ide 1:3 200000 /etc/fb.modes

24541 bytes read
=> md 200000
00200000: 230a2320 20205361 6d706c65 20766964    #.#   Sample vid
00200010: 656f206d 6f646573 0a23200a 23202020    eo modes.# .#
00200020: 54686573 65206461 74612061 72652062    These data are b
00200030: 61736564 206f6e20 74686520 43525443    ased on the CRTC
00200040: 20706172 616d6574 65727320 696e0a23     parameters in.#
00200050: 200a2320 20202020 20204d61 63683634     .#       Mach64
00200060: 2050726f 6772616d 6d657227 73204775     Programmer's Gu
…
```

6.7.3 Booting from an IDE Device

The following example shows how Linux could be booted from a CompactFlash card.

First a Linux system (kernel + root Filesystem) must be copied to the CF card. To do this boot the Linux NFS system (**run net_nfs**). Then two partitions have to be created on the CF card (one for the kernel and one for the root Filesystem).

Hint: All Data on the CF card will be lost!

```
target# fdisk /dev/hda

Command (m for help): p

Disk /dev/hda: 128 MB, 128188416 bytes
8 heads, 32 sectors/track, 978 cylinders
Units = cylinders of 256 * 512 = 131072 bytes

   Device Boot    Start       End    Blocks   Id  System
/dev/hda1             1       978    125168    4  FAT16 <32M

Command (m for help): d
Selected partition 1

Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
Partition number (1-4): 1
First cylinder (1-978, default 1):
Using default value 1
Last cylinder or +size or +sizeM or +sizeK (1-978, default 978): +4M

Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
```

```
Partition number (1-4): 2
First cylinder (33-978, default 33):
Using default value 33
Last cylinder or +size or +sizeM or +sizeK (33-978, default 978):
Using default value 978

Command (m for help): p

Disk /dev/hda: 128 MB, 128188416 bytes
8 heads, 32 sectors/track, 978 cylinders
Units = cylinders of 256 * 512 = 131072 bytes

   Device Boot    Start       End    Blocks   Id  System
/dev/hda1             1        32      4080   83  Linux
/dev/hda2            33       978    121088   83  Linux

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
```

Now the kernel is copied to the first partition (before that the kernel image must be copied from the TQ-CD to the NFS filesystem:

```
host# cp <path on cd>/uImage_tqm5200_2005-05-17_usb_mass_storage
/opt/eldk/ppx_6xx/images/
```

On the target:

```
target# cp /images/uImage_tqm5200_2005-05-17_usb_mass_storage /dev/hda1
```

Then the root filesystem is created on partition 2 of the CF card:

```
target# mkfs.ext2 /dev/hda2
mke2fs 1.27 (8-Mar-2002)
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
30360 inodes, 121088 blocks
6054 blocks (5.00%) reserved for the super user
First data block=1
15 block groups
8192 blocks per group, 8192 fragments per group
2024 inodes per group
Superblock backups stored on blocks:
        8193, 24577, 40961, 57345, 73729

Writing inode tables: done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 29 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
target# cp /images/ramdisk_image.gz /images/ramdisk_image.bak.gz
target# gunzip /images/ramdisk_image.bak.gz
```

Hint: If the device node /dev/loop0 doesn't exist, create it with the **mknod** tool:

```
target# mknod /dev/loop0 b 7 0

target# mount -o loop /images/ramdisk_image.bak /mnt/tmp/
target# ls -l /mnt/tmp/
total 14
```


LIN5200.SWM.105.doc

```
drwxr-xr-x    2 1543     1543         2048 Nov 10  2004 bin
drwxr-xr-x    2 root     root         3072 Nov 10  2004 dev
drwxr-xr-x    5 1543     1543         1024 Nov 10  2004 etc
drwxr-xr-x    2 1543     1543         1024 Nov 10  2004 ftp
drwxr-xr-x    2 1543     1543         1024 Nov 10  2004 home
drwxr-xr-x    3 1543     1543         1024 Nov 10  2004 lib
drwxr-xr-x    2 1543     1543         1024 Nov 10  2004 proc
drwxr-xr-x    2 1543     1543         1024 Nov 10  2004 sbin
drwxr-xr-x    2 1543     1543         1024 Nov 10  2004 tmp
drwxr-xr-x    4 1543     1543         1024 Nov 10  2004 usr
drwxr-xr-x    4 1543     1543         1024 Nov 10  2004 var
target# mount /dev/hda2 /mnt/hda2
target# cp -a /mnt/tmp/* /mnt/hda2/
target# ls -l /mnt/hda2/
total 26
drwxr-xr-x    2 1543     1543         2048 Nov 10  2004 bin
drwxr-xr-x    2 root     root         3072 Nov 10  2004 dev
drwxr-xr-x    5 1543     1543         1024 Nov 10  2004 etc
drwxr-xr-x    2 1543     1543         1024 Nov 10  2004 ftp
drwxr-xr-x    2 1543     1543         1024 Nov 10  2004 home
drwxr-xr-x    3 1543     1543         1024 Nov 10  2004 lib
drwx------    2 root     root        12288 Dec 31 19:05 lost+found
drwxr-xr-x    2 1543     1543         1024 Nov 10  2004 proc
drwxr-xr-x    2 1543     1543         1024 Nov 10  2004 sbin
drwxr-xr-x    2 1543     1543         1024 Nov 10  2004 tmp
drwxr-xr-x    4 1543     1543         1024 Nov 10  2004 usr
drwxr-xr-x    4 1543     1543         1024 Nov 10  2004 var
target# mount
/dev/nfs on / type nfs (rw)
none on /proc type proc (rw)
usbdevfs on /proc/bus/usb type usbdevfs (rw)
/images/ramdisk_image.bak on /mnt/tmp type ext2 (rw,loop=/dev/loop0)
/dev/hda2 on /mnt/hda2 type ext2 (rw)
target# umount /mnt/hda2
umtarget# umount /mnt/tmp/
target# rm /images/ramdisk_image.bak
target#
```

After that run U-Boot:

```
=> set hddargs 'setenv bootargs root=/dev/hda2 ro'
=> set hdd_hdd 'diskboot 200000 0:1; run hddargs addip addcons; bootm 200000'
=> save
Saving Environment to Flash...
Un-Protected 1 sectors
Un-Protected 1 sectors
Erasing Flash...
. done
Erased 1 sectors
Writing to Flash... done
Protected 1 sectors
Protected 1 sectors
=> print hddargs
hddargs=setenv bootargs root=/dev/hda2 ro
=> print hdd_hdd
hdd_hdd=diskboot 200000 0:1; run hddargs addip addcons; bootm 200000
```

Now Linux could be booted from the CF card:

```
=> run hdd_hdd

Loading from IDE device 0, partition 1: Name: hda1
  Type: U-Boot
   Image Name:   Linux-2.4.25
```

```
   Created:      2005-05-17  12:00:07 UTC
   Image Type:   PowerPC Linux Kernel Image (gzip compressed)
   Data Size:    1057337 Bytes =  1 MB
   Load Address: 00000000
   Entry Point:  00000000
## Booting image at 00200000 ...
   Image Name:   Linux-2.4.25
   Created:      2005-05-17  12:00:07 UTC
   Image Type:   PowerPC Linux Kernel Image (gzip compressed)
   Data Size:    1057337 Bytes =  1 MB
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK
   Uncompressing Kernel Image ... OK
Memory BAT mapping: BAT2=128Mb, BAT3=0Mb, residual: 0Mb
Linux version 2.4.25 (mkr@s020403) (gcc version 3.3.3 (DENX ELDK 3.1 3.3.3-8)) #
2 Tue May 17 13:59:09 CEST 2005
On node 0 totalpages: 32768
zone(0): 32768 pages.
zone(1): 0 pages.
zone(2): 0 pages.
Kernel command line: root=/dev/hda2 ro ip=172.20.5.73:172.20.5.13::::eth0:off pa
nic=1 console=tty0 console=ttyS0,115200
Console: colour dummy device 80x25
Calibrating delay loop... 263.78 BogoMIPS
Memory: 126548k available (1764k kernel code, 600k data, 100k init, 0k highmem)
Dentry cache hash table entries: 16384 (order: 5, 131072 bytes)
Inode cache hash table entries: 8192 (order: 4, 65536 bytes)
Mount cache hash table entries: 512 (order: 0, 4096 bytes)
Buffer cache hash table entries: 8192 (order: 3, 32768 bytes)
Page-cache hash table entries: 32768 (order: 5, 131072 bytes)
POSIX conformance testing by UNIFIX
PCI: Probing PCI hardware
PCI: Cannot allocate resource region 0 of device 00:1a.0
Linux NET4.0 for Linux 2.4
Based upon Swansea University Computer Society NET3.039
Initializing RT netlink socket
Starting kswapd
Journalled Block Device driver loaded
Installing knfsd (copyright (C) 1996 okir@monad.swb.de).
JFFS2 version 2.2. (NAND) (C) 2001-2003 Red Hat, Inc.
Console: switching to colour frame buffer device 80x60
Silicon Motion, Inc. Voyager Init Complete.
PS/2 Multiplexer driver
Keyboard: not found
Mouse: not found
initialize_kbd: Keyboard reset failed, no ACK
ps2mult.c: keyboard command not acknowledged
Detected PS/2 Mouse Port.
pty: 256 Unix98 ptys configured
keyboard: Timeout - AT keyboard not present?(ed)
ps2mult.c: keyboard command not acknowledged
keyboard: Timeout - AT keyboard not present?(f4)
ttyS0 on PSC1
ttyS1 on PSC2
ttyS2 on PSC3
SRAM_DRV: initialized
RAMDISK driver initialized: 16 RAM disks of 10240K size 1024 blocksize
loop: loaded (max 8 devices)
Uniform Multi-Platform E-IDE driver Revision: 7.00beta4-2.4
ide: Assuming 33MHz system bus speed for PIO modes; override with idebus=xx
Port Config is: 0x81550014
ipb=132MHz, set clock period to 7
```

```
GPIO config: 81550014
ATA invalid: 01000000
ATA hostcnf: 03000000
ATA pio1   : 100a0a00
ATA pio2   : 02040600
XLB Arb cnf: 8000a366
mpc5xxx_ide: Setting up IDE interface ide0...
Probing IDE interface ide0...
hda: TOSHIBA THNCF128MBA, CFA DISK drive
ide0 at 0xf0003a60-0xf0003a67,0xf0003a5c on irq 45
hda: attached ide-disk driver.
hda: 250368 sectors (128 MB) w/2KiB Cache, CHS=978/8/32
Partition check:
 hda: hda1 hda2
SCSI subsystem driver Revision: 1.00
kmod: failed to exec /sbin/modprobe -s -k scsi_hostadapter, errno = 2
init_tqm5200_mtd: chip probing count 0
TQM5200-0: Found 2 x16 devices at 0x0 in 32-bit bank
 Amd/Fujitsu Extended Query Table at 0x0040
TQM5200-0: CFI does not contain boot bank location. Assuming top.
number of CFI chips: 1
cfi_cmdset_0002: Disabling erase-suspend-program due to code brokenness.
init_tqm5200_mtd: bank 1, name: TQM5200-0, size: 67108864 bytes
init_tqm5200_mtd: -22 command line partitions on bank 0
TQM5200 flash bank 0: Using static image partition definition
Creating 6 MTD partitions on "TQM5200-0":
0x00000000-0x000a0000 : "firmware"
0x000a0000-0x00200000 : "kernel"
0x00200000-0x00400000 : "initrd"
0x00400000-0x00800000 : "small-fs"
0x00800000-0x01800000 : "big-fs"
0x01800000-0x02000000 : "misc"
usb.c: registered new driver usbdevfs
usb.c: registered new driver hub
host/usb-ohci.c: USB OHCI at membase 0xf0001000, IRQ 44
host/usb-ohci.c: usb-0, Built-In ohci
usb.c: new USB bus registered, assigned bus number 1
hub.c: USB hub found
hub.c: 1 port detected
usb.c: registered new driver hiddev
usb.c: registered new driver hid
hid-core.c: v1.8.1 Andreas Gal, Vojtech Pavlik <vojtech@suse.cz>
hid-core.c: USB HID support drivers
Initializing USB Mass Storage driver...
usb.c: registered new driver usb-storage
USB Mass Storage support registered.
mice: PS/2 mouse device common for all mice
NET4: Linux TCP/IP 1.0 for NET4.0
eth0: Phy @ 0x0, type LXT971 (0x001378e2)
IP Protocols: ICMP, UDP, TCP, IGMP
IP: routing cache hash table of 1024 buckets, 8Kbytes
TCP: Hash tables configured (established 8192 bind 16384)
eth0: config: auto-negotiation on, 100FDX, 100HDX, 10FDX, 10HDX.
IP-Config: Guessing netmask 255.255.0.0
IP-Config: Complete:
      device=eth0, addr=172.20.5.73, mask=255.255.0.0, gw=255.255.255.255,
     host=172.20.5.73, domain=, nis-domain=(none),
     bootserver=172.20.5.13, rootserver=172.20.5.13, rootpath=
NET4: Unix domain sockets 1.0/SMP for Linux NET4.0.
 hda: hda1 hda2
 hda: hda1 hda2
 hda: hda1 hda2
 hda: hda1 hda2
```

```
VFS: Mounted root (ext2 filesystem) readonly.
Freeing unused kernel memory: 100k init
/etc/mtab: cannot create (Read-only file system)
open: Read-only file system
eth0: status: link up, 100 Mbps Full Duplex, auto-negotiation complete.


BusyBox v0.60.5 (2004.11.10-09:40+0000) Built-in shell (msh)
Enter 'help' for a list of built-in commands.

# ### Application running ...
mount
rootfs on / type rootfs (rw)
/dev/hda on / type ext2 (ro)
/proc on /proc type proc (rw)
```

Note that "/" is mounted "read only" (ro). This is done, because flash devices only allow a limited number of writes cycles. If you want to change files or create new files on the CF card, "/" could be remounted "read-write" after booting. With the mount option "noatime" the access timestamps are not updated when a file is read. This significantly reduces the number of write cycles to the CF card and should always be set for flash devices:

```
# mount /dev/hda2 / -o remount,noatime
EXT2-fs warning: mounting unchecked fs, running e2fsck is recommended
# mount
rootfs on / type rootfs (rw)
/dev/hda on / type ext2 (rw,noatime)
/proc on /proc type proc (rw)
```

6.8 USB Subsystem

U-Boot supports commands to get information about the connected USB devices and to access USB mass storage devices.

The commands of the USB subsystem are started with **usb** followed by a subcommand:

```
=> help usb
usb reset - reset (rescan) USB controller
usb stop [f]  - stop USB [f]=force stop
usb tree  - show USB device tree
usb info [dev] - show available USB devices
usb scan  - (re-)scan USB bus for storage devices
usb device [dev] - show or set current USB storage device
usb part [dev] - print partition table of one or all USB storage devices
usb read addr blk# cnt - read `cnt' blocks starting at block `blk#'
    to memory address `addr'
```

6.8.1 Getting Information about USB Devices

After connecting an USB Device an **usb reset** command must be issued to initialize the USB device (this is because U-Boot does not support USB hot plugging und thus does not know when a new USB hardware is connected).

```
=> usb reset
(Re)start USB...
USB:   scanning bus for devices... 2 USB Device(s) found
       scanning bus for storage devices... 1 Storage Device(s) found
```

Now the commands **usb tree** and **usb info** could be used to show some information about the connected USB device:

LIN5200.SWM.105.doc

```
=> usb info
1: Hub,  USB Revision 1.10
 -  OHCI Root Hub
 - Class: Hub
 - PacketSize: 8  Configurations: 1
 - Vendor: 0x0000  Product 0x0000 Version 0.0
   Configuration: 1
   - Interfaces: 1 Self Powered 0mA
     Interface: 0
     - Alternate Settings 0, Endpoints: 1
     - Class Hub
     - Endpoint 1 In Interrupt MaxPacket 2 Interval 255ms

2: Mass Storage,  USB Revision 1.10
 - USB Solid state disk 403210023ED2B963
 - Class: (from Interface) Mass Storage
 - PacketSize: 64  Configurations: 1
 - Vendor: 0x0ea0  Product 0x6803 Version 1.0
   Configuration: 1
   - Interfaces: 1 Bus Powered 100mA
     Interface: 0
     - Alternate Settings 0, Endpoints: 3
     - Class Mass Storage, Transp. SCSI, Bulk only
     - Endpoint 1 In Bulk MaxPacket 64
     - Endpoint 2 Out Bulk MaxPacket 64
     - Endpoint 3 In Interrupt MaxPacket 2 Interval 1ms

=> usb tree

Device Tree:
  1  Hub (12MBit/s, 0mA)
  |   OHCI Root Hub
  |
  +-2  Mass Storage (12MBit/s, 100mA)
       USB Solid state disk 403210023ED2B963
```

6.8.2 Accessing USB Mass Storage Devices

Note: The following description is for the U-Boot versions **u-boot_tqm5200_2005-11-04_stk52xx.100.bin** or **u-boot_tqm5200_2005-11-04_stk52xx.200.bin** and later. In older versions some commands are slightly different (e. g. **usb scan** is used instead of **usb storage**).

The **usb storage** command shows details of recognized mass storage devices:

```
=> usb storage
  Device 0: Vendor: JetFlash Prod.: 32MB             Rev: 1.11
            Type: Removable Hard Disk
            Capacity: 31.5 MB = 0.0 GB (64512 x 512)
```

Now the partition table could be read with the **usb part** command:

```
=> usb part
print_part of 0

Partition Map for USB device 0  --   Partition Type: DOS

Partition     Start Sector     Num Sectors     Type
    1                   32           64416       4

print_part of 1
## Unknown partition table
```

```
print_part of 2
## Unknown partition table

print_part of 3
## Unknown partition table

print_part of 4
## Unknown partition table
```

The usb read command could be used to read data from an USB mass storage device:

```
=> usb read 200000 0 1

USB read: device 0 block # 0, count 1 ... 1 blocks read: OK
```

This reads one block (= 512 bytes for this device) of data beginning at block #0 and copies it to the address 0x2000000. On storage devices the first block (sector) usually contains the master boot record (MBR) with the boot loader and the partition table (starting at offset 0x1BE)

```
=> md 200000
00200000: fa33c08e d0bc007c 8bf45007 501ffbfc    .3.....|..P.P...
00200010: bf0006b9 0001f2a5 ea1d0600 00bebe07    ................
00200020: b304803c 80740e80 3c00751c 83c610fe    ...<.t..<.u.....
00200030: cb75efcd 188b148b 4c028bee 83c610fe    .u......L.......
00200040: cb741a80 3c0074f4 be8b06ac 3c00740b    .t..<.t.....<.t.
00200050: 56bb0700 b40ecd10 5eebf0eb febf0500    V.......^.......
00200060: bb007cb8 010257cd 135f730c 33c0cd13    ..|...W.._s.3...
00200070: 4f75edbe a306ebd3 bec206bf fe7d813d    Ou..........}.=
00200080: 55aa75c7 8bf5ea00 7c000049 6e76616c    U.u.....|..Inval
00200090: 69642070 61727469 74696f6e 20746162    id partition tab
002000a0: 6c650045 72726f72 206c6f61 64696e67    le.Error loading
002000b0: 206f7065 72617469 6e672073 79737465     operating syste
002000c0: 6d004d69 7373696e 67206f70 65726174    m.Missing operat
002000d0: 696e6720 73797374 656d0000 00000000    ing system......
002000e0: 00000000 00000000 00000000 00000000    ................
002000f0: 00000000 00000000 00000000 00000000    ................
=>
00200100: 00000000 00000000 00000000 00000000    ................
00200110: 00000000 00000000 00000000 00000000    ................
00200120: 00000000 00000000 00000000 00000000    ................
00200130: 00000000 00000000 00000000 00000000    ................
00200140: 00000000 00000000 00000000 00000000    ................
00200150: 00000000 00000000 00000000 00000000    ................
00200160: 00000000 00000000 00000000 00000000    ................
00200170: 00000000 00000000 00000000 00000000    ................
00200180: 00000000 00000000 00000000 00000000    ................
00200190: 00000000 00000000 00000000 00000000    ................
002001a0: 00000000 00000000 00000000 00000000    ................
002001b0: 00000000 00000000 00000000 00008001    ................
002001c0: 01000401 e0ee2000 0000a0fb 00000000    ...... .........
002001d0: 00000000 00000000 00000000 00000000    ................
002001e0: 00000000 00000000 00000000 00000000    ................
002001f0: 00000000 00000000 00000000 000055aa    ..............U.
```

Hint: Writing to USB devices is not supported in U-Boot.

Because working with the **`usb read`** command is not very comfortable if you want to read whole files from a filesystem, U-Boot implements commands to access the following filesystems: FAT, ext2, VFAT. cramfs. reiser and jffs2.

The **`fatls`** command lists the contents of a FAT filesystem:

LIN5200.SWM.105.doc

```
=> help fatls
fatls <interface> <dev[:part]> [directory]
    - list files from 'dev' on 'interface' in a 'directory'

=> fatls usb 0:1
            microwindows/
         0  testfile
    782828  kernel.img
  16578134  uramdisk_tqm5200_16mb_net-test_mkr.img

3 file(s), 1 dir(s)

=> fatls usb 0:1 microwindows
            ./
            ../
            bin/
            lib/
            tuxchess/

0 file(s), 5 dir(s)
```

The command **`ext2ls`** could be used similarly for ext2 filesystems.

With the **`fatload`** command files from a FAT filesytem could be loaded to memory:

```
=> help fatload
fatload <interface> <dev[:part]>  <addr> <filename> [bytes]
    - load binary file 'filename' from 'dev' on 'interface'
      to address 'addr' from dos filesystem

=> fatload usb 0:1 200000 kernel.img
reading kernel.img
.................................................................................

782828 bytes read
=> imi 200000

## Checking Image at 00200000 ...
   Image Name:   ARM Linux-2.4.18
   Created:      2005-03-18   8:10:32 UTC
   Image Type:   ARM Linux Kernel Image (gzip compressed)
   Data Size:    782764 Bytes = 764.4 kB
   Load Address: 0c008000
   Entry Point:  0c008000
   Verifying Checksum ... OK
```

The command **`ext2load`** could be used similarly for ext2 filesystems.

6.8.3 Booting from an USB Stick

With the **`usbboot`** command a Linux system could be booted from an USB stick.

For this purpose the USB stick must be prepared. This could be done under Linux on the target running a NFS system. It is important, that support for USB mass storage devices is activated in the kernel (for instance the kernel image **`uImage_tqm5200_2005-05-17_usb_mass_storage`** could be used).

```
target# cat /proc/scsi/usb-storage-0/0
   Host scsi0: usb-storage
       Vendor: USB
      Product: Solid state disk
Serial Number: 403210023ED2C367
```

```
     Protocol: Transparent SCSI
    Transport: Bulk
         GUID: 0ea06803403210023ed2c367
     Attached: Yes
target# fdisk /dev/sda

Command (m for help): p

Disk /dev/sda: 33 MB, 33030144 bytes
2 heads, 32 sectors/track, 1008 cylinders
Units = cylinders of 64 * 512 = 32768 bytes

   Device Boot    Start       End    Blocks   Id  System
/dev/sda1   *         1      1007     32208    4  FAT16 <32M

Command (m for help): d
Selected partition 1

Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
Partition number (1-4): 1
First cylinder (1-1008, default 1):
Using default value 1
Last cylinder or +size or +sizeM or +sizeK (1-1008, default 1008): +4M

Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
Partition number (1-4): 2
First cylinder (124-1008, default 124):
Using default value 124
Last cylinder or +size or +sizeM or +sizeK (124-1008, default 1008):
Using default value 1008

Command (m for help): p

Disk /dev/sda: 33 MB, 33030144 bytes
2 heads, 32 sectors/track, 1008 cylinders
Units = cylinders of 64 * 512 = 32768 bytes

   Device Boot    Start       End    Blocks   Id  System
/dev/sda1             1       123      3920   83  Linux
/dev/sda2           124      1008     28320   83  Linux

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
SCSI device sda: 64512 512-byte hdwr sectors (33 MB)
sda: Write Protect is off
SCSI device sda: 64512 512-byte hdwr sectors (33 MB)
sda: Write Protect is off
Syncing disks.
target# cp /images/uImage_tqm5200_2005-05-17_usb_mass_storage /dev/sda1
target# cp /images/uRamdisk /dev/sda2
```

In U-Boot some environment variables have to be set:

```
=> set usb_self 'run usbload; run ramargs addip addcons; bootm 200000 400000'
=> set addcons 'setenv bootargs $(bootargs) console=tty0 console=ttyS0,$(baudrat
e)'
=> set usbload 'usb reset; usbboot 200000 0:1; usbboot 400000 0:2'
=> print usb_self
usb_self=run usbload; run ramargs addip addcons; bootm 200000 400000
=> print usbload
usbload=usb reset; usb scan; usbboot 200000 0:1; usbboot 400000 0:2
=> print addcons
addcons=setenv bootargs $(bootargs) console=tty0 console=ttyS0,$(baudrate)
```

Now Linux could be booted from USB stick:

```
=> run usb_self
(Re)start USB...
USB:   scanning bus for devices... 2 USB Devices found
Scan for storage device:
       scanning bus for storage devices...
  Device 0: Vendor: JetFlash Prod.: 32MB             Rev: 1.11
            Type: Removable Hard Disk
            Capacity: 31.5 MB = 0.0 GB (64512 x 512)

Loading from USB device 0, partition 1: Name: usbda1
  Type: U-Boot
   Image Name:   Linux-2.4.25
   Created:      2005-05-17  12:00:07 UTC
   Image Type:   PowerPC Linux Kernel Image (gzip compressed)
   Data Size:    1057337 Bytes =  1 MB
   Load Address: 00000000
   Entry Point:  00000000
.................................................................................
.......................

Loading from USB device 0, partition 2: Name: usbda2
  Type: U-Boot
   Image Name:   Simple Embedded Linux Framework
   Created:      2004-11-10   9:45:02 UTC
   Image Type:   PowerPC Linux RAMDisk Image (gzip compressed)
   Data Size:    1581051 Bytes =  1.5 MB
   Load Address: 00000000
   Entry Point:  00000000
.................................................................................
.....................................................................
## Booting image at 00200000 ...
   Image Name:   Linux-2.4.25
   Created:      2005-05-17  12:00:07 UTC
   Image Type:   PowerPC Linux Kernel Image (gzip compressed)
   Data Size:    1057337 Bytes =  1 MB
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK
   Uncompressing Kernel Image ... OK
## Loading RAMDisk Image at 00400000 ...
   Image Name:   Simple Embedded Linux Framework
   Created:      2004-11-10   9:45:02 UTC
   Image Type:   PowerPC Linux RAMDisk Image (gzip compressed)
   Data Size:    1581051 Bytes =  1.5 MB
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK
   Loading Ramdisk to 07db8000, end 07f39ffb ... OK
```

```
Memory BAT mapping: BAT2=128Mb, BAT3=0Mb, residual: 0Mb
Linux version 2.4.25 (mkr@s020403) (gcc version 3.3.3 (DENX ELDK 3.1 3.3.3-8)) #
2 Tue May 17 13:59:09 CEST 2005
On node 0 totalpages: 32768
zone(0): 32768 pages.
zone(1): 0 pages.
zone(2): 0 pages.
Kernel command line: root=/dev/ram rw ip=172.20.5.73:172.20.5.13::::eth0:off pan
ic=1 console=tty0 console=ttyS0,115200
Console: colour dummy device 80x25
Calibrating delay loop... 263.78 BogoMIPS
Memory: 125004k available (1764k kernel code, 600k data, 100k init, 0k highmem)
Dentry cache hash table entries: 16384 (order: 5, 131072 bytes)
Inode cache hash table entries: 8192 (order: 4, 65536 bytes)
Mount cache hash table entries: 512 (order: 0, 4096 bytes)
Buffer cache hash table entries: 8192 (order: 3, 32768 bytes)
Page-cache hash table entries: 32768 (order: 5, 131072 bytes)
POSIX conformance testing by UNIFIX
PCI: Probing PCI hardware
PCI: Cannot allocate resource region 0 of device 00:1a.0
Linux NET4.0 for Linux 2.4
Based upon Swansea University Computer Society NET3.039
Initializing RT netlink socket
Starting kswapd
Journalled Block Device driver loaded
Installing knfsd (copyright (C) 1996 okir@monad.swb.de).
JFFS2 version 2.2. (NAND) (C) 2001-2003 Red Hat, Inc.
Console: switching to colour frame buffer device 80x60
Silicon Motion, Inc. Voyager Init Complete.
PS/2 Multiplexer driver
Keyboard: not found
Mouse: not found
initialize_kbd: Keyboard reset failed, no ACK
ps2mult.c: keyboard command not acknowledged
Detected PS/2 Mouse Port.
pty: 256 Unix98 ptys configured
keyboard: Timeout - AT keyboard not present?(ed)
ps2mult.c: keyboard command not acknowledged
keyboard: Timeout - AT keyboard not present?(f4)
ttyS0 on PSC1
ttyS1 on PSC2
ttyS2 on PSC3
SRAM_DRV: initialized
RAMDISK driver initialized: 16 RAM disks of 10240K size 1024 blocksize
loop: loaded (max 8 devices)
Uniform Multi-Platform E-IDE driver Revision: 7.00beta4-2.4
ide: Assuming 33MHz system bus speed for PIO modes; override with idebus=xx
Port Config is: 0x91551004
ipb=132MHz, set clock period to 7
GPIO config: 91551004
ATA invalid: 01000000
ATA hostcnf: 03000000
ATA pio1   : 100a0a00
ATA pio2   : 02040600
XLB Arb cnf: 8000a366
mpc5xxx_ide: Setting up IDE interface ide0...
Probing IDE interface ide0...
hda: TOSHIBA THNCF128MBA, CFA DISK drive
ide0 at 0xf0003a60-0xf0003a67,0xf0003a5c on irq 45
hda: attached ide-disk driver.
hda: 250368 sectors (128 MB) w/2KiB Cache, CHS=978/8/32
Partition check:
 hda: hda1 hda2
```

```
SCSI subsystem driver Revision: 1.00
kmod: failed to exec /sbin/modprobe -s -k scsi_hostadapter, errno = 2
init_tqm5200_mtd: chip probing count 0
TQM5200-0: Found 2 x16 devices at 0x0 in 32-bit bank
 Amd/Fujitsu Extended Query Table at 0x0040
TQM5200-0: CFI does not contain boot bank location. Assuming top.
number of CFI chips: 1
cfi_cmdset_0002: Disabling erase-suspend-program due to code brokenness.
init_tqm5200_mtd: bank 1, name: TQM5200-0, size: 67108864 bytes
init_tqm5200_mtd: -22 command line partitions on bank 0
TQM5200 flash bank 0: Using static image partition definition
Creating 6 MTD partitions on "TQM5200-0":
0x00000000-0x000a0000 : "firmware"
0x000a0000-0x00200000 : "kernel"
0x00200000-0x00400000 : "initrd"
0x00400000-0x00800000 : "small-fs"
0x00800000-0x01800000 : "big-fs"
0x01800000-0x02000000 : "misc"
usb.c: registered new driver usbdevfs
usb.c: registered new driver hub
host/usb-ohci.c: USB OHCI at membase 0xf0001000, IRQ 44
host/usb-ohci.c: usb-0, Built-In ohci
usb.c: new USB bus registered, assigned bus number 1
hub.c: USB hub found
hub.c: 1 port detected
usb.c: registered new driver hiddev
usb.c: registered new driver hid
hid-core.c: v1.8.1 Andreas Gal, Vojtech Pavlik <vojtech@suse.cz>
hid-core.c: USB HID support drivers
Initializing USB Mass Storage driver...
usb.c: registered new driver usb-storage
USB Mass Storage support registered.
mice: PS/2 mouse device common for all mice
NET4: Linux TCP/IP 1.0 for NET4.0
eth0: Phy @ 0x0, type LXT971 (0x001378e2)
IP Protocols: ICMP, UDP, TCP, IGMP
IP: routing cache hash table of 1024 buckets, 8Kbytes
TCP: Hash tables configured (established 8192 bind 16384)
eth0: config: auto-negotiation on, 100FDX, 100HDX, 10FDX, 10HDX.
IP-Config: Guessing netmask 255.255.0.0
IP-Config: Complete:
      device=eth0, addr=172.20.5.73, mask=255.255.0.0, gw=255.255.255.255,
     host=172.20.5.73, domain=, nis-domain=(none),
     bootserver=172.20.5.13, rootserver=172.20.5.13, rootpath=
NET4: Unix domain sockets 1.0/SMP for Linux NET4.0.
RAMDISK: Compressed image found at block 0
Freeing initrd memory: 1543k freed
VFS: Mounted root (ext2 filesystem).
Freeing unused kernel memory: 100k init
hub.c: new USB device 0-1, assigned address 2
scsi0 : SCSI emulation for USB Mass Storage devices
  Vendor: JetFlash  Model: 32MB              Rev: 1.11
  Type:   Direct-Access                      ANSI SCSI revision: 02
Attached scsi removable disk sda at scsi0, channel 0, id 0, lun 0
SCSI device sda: 64512 512-byte hdwr sectors (33 MB)
sda: Write Protect is off
 sda: sda1 sda2
eth0: status: link up, 100 Mbps Half Duplex, auto-negotiation complete.


BusyBox v0.60.5 (2004.11.10-09:40+0000) Built-in shell (msh)
Enter 'help' for a list of built-in commands.
```

```
# ### Application running ...

# cat /proc/version
Linux version 2.4.25 (mkr@s020403) (gcc version 3.3.3 (DENX ELDK 3.1 3.3.3-8)) #
2 Tue May 17 13:59:09 CEST 2005
#
```

Hint: During booting of the kernel, the system seems to hang, but after a time of about 40 s, the boot process is continued. The reason for this behavior hasn't been analyzed, yet..

6.9 Backlight (TB5200)

On the TB5200 baseboard there is a switchable 12 V output for supplying of a backlight inverter.

With the **backlight** command this output could be turned on or off.

```
=> help backlight
backlight on/off
```

6.10 LM75 temperature sensor (TQM5200S)

The LM75 temperature sensor on the TQM5200S could be queried via a I2C command:

```
=> imd 48 0.1 2
0000: 20 ff
```

The first byte of the returned value (20 hex) is the current measured temperature in degrees Centigrade. The returned value is presented as hexadecimal two's complement.

For positive Temperatures the value could directly be converted from hexadecimal to decimal:

20 hex -> 32 dec -> Temperature: 32°C.

For negative Temperatures (the MSB is set) the two's complement had to be calculated and then the result converted from hexadecimal to decimal:

FC hex -> 11111100 bin -> two's complement (invert and add 1) -> 00000100 bin -> 4 dec -> Temperature: -4°C

7. Working with Linux

This chapter describes the use of some Linux drivers for the TQM5200 (and the STK52xx) and gives some tips on developing with Linux. For a general documentation on using Linux look at [2].

7.1 Compiling Applications

There are two ways for compiling Linux applications:

- on the Linux Host with the Cross Compiler
- on the target with the native compiler

7.1.1 Compiling with the Cross Compiler

Target applications could be compiled on the Linux host. Because the architecture of the target is not the same as the architecture on the Linux host, you have to use a cross compiler.

If you've installed ELDK, cross compiling for the target is very easy. Make sure that the environment variables **$PATH** is set up properly. **$PATH** must contain the path where you installed ELDK.

```
host# echo $PATH
/opt/eldk/usr/bin:/opt/eldk/bin:/sbin:/usr/sbin:/bin:/usr/bin:/usr/X11R6/bin:/usr/loc
al/bin
```

With a name prefix to **gcc** the correct cross compiler is selected (**ppc_6xx-gcc** is a symbolic link to the correct cross compiler):

```
host# ppc_6xx-gcc -Wall -o hello_world hello_world.c
host# ll hello_world
-rwxrwxr-x    1 fpe      fpe         15013 May 11 09:49 hello_world
```

The resulting file is not executable on the Linux host, because it is compiled for the target architecture.

```
host# file hello_world
hello_world: ELF 32-bit MSB executable, PowerPC or cisco 4500, version 1 (SYSV),
dynamically linked (uses shared libs), not stripped
host# ./hello_world
bash: ./hello_world: cannot execute binary file
```

To run the demo copy the file to the target file system (for instance to the via NFS mounted root file system) and start it on the target:

```
target# ./hello_world
Hello World
```

7.1.2 Compiling with the native Target Compiler

Instead of using the Linux host for compiling, applications could be compiled directly on the target. Thus the native target compiler is used.

To compile the "hello_world" demo application start Linux with a root file system mounted via NFS (**run net_nfs**) and copy the **hello_world.c** source to the mounted file system:

```
target# gcc -Wall -o hello_world hello_world.c
target# ll hello_world
-rwxr-xr-x    1 root     root        14926 Dec 31 19:25 hello_world
target# ./hello_world
Hello World
```

7.2 SM501 “voyager” Frame-buffer Driver

The **voyager** driver is a frame-buffer device driver for the SM501 graphic controller. Because it is a frame-buffer driver no hardware acceleration is supported.

Hint: The frame-buffer driver could only be used on TQM5200 modules with on-board graphic controller. There are some module variants which don’t have a graphic controller. Make sure not to activate the driver on this variants during kernel configuration.

The driver prints a message during kernel boot time if it is loaded successfully:

```
target# dmesg | grep -i voyager
Silicon Motion, Inc. Voyager Init Complete.
```

The first frame-buffer device node has a major number of 29 and minor number 0. If it doesn’t exist you have to create it before the driver could be used:

```
target# mknod /dev/fb0 c 29 0
target# ls -l /dev/fb0
crw-------    1 root     root      29,   0 Dec  9  2004 /dev/fb0
```

The frame-buffer device could now be accessed over the special device file **/dev/fb0**. **/dev/fb** normally is a symbolic link to **/dev/fb0**.

A more detailed description of the frame-buffer interface could be found here [3].

7.2.1 Kernel Bootparameter “video”

With the kernel boot parameter **video=voyager** user specific parameters could be passed to the **voyager** driver.

The following parameters are supported currently:

```
video=voyager:fp
```

This is the default configuration (and also used if the video kernel boot parameter is omitted completely). As output device the FP (flat panel) is used. In addition to that the FP output signal is copied to the CRT interface.

```
video=voyager:crt
```

With this configuration only the CRT interface is used as output device. No data is send to the FP interface.

```
video=voyager:bpp=<bpp>
```

With <bpp> the used color depth (bits per pixel) could be set. Currently only 32 bpp are recommended.

Hint: In Linux the same output device (CRT or FP) must be used as in U-Boot. U-Boot uses the FP device (with data copied to the CRT interface). Thus if in U-Boot the graphic console is configured, in Linux the FP configuration has to be used. Otherwise the graphic device could hang-up. The CRT option in Linux only can be used if the graphic console in U-Boot is *not* configured.

Example bootargs:

```
=> print bootargs
bootargs=root=/dev/nfs rw nfsroot=172.20.5.13:/opt/eldk/ppx_6xx/ ip=172.
20.5.73:172.20.5.13::::eth0:off panic=1 video=voyager:fp
```

7.2.2 Changing video Parameters during Run-time with "fbset"

During run-time the frame-buffer driver could be used to change the video settings like display timing or resolution. The voyager driver provides a well defined set of **`ioctl()`** system calls to change the video settings [3].

The tool **`fbset`** could be used to show and modify frame buffer device settings. A version of **`fbset`** is provided on the TQ-CD. Copy the file to the **`/usr/sbin/`** directory on the target file system. The file with the video mode database (**`fb.modes`**) has to be copied to **`/etc/`**.

If **`fbset`** is called without parameters the current video settings are printed:

```
target# fbset

mode "640x480-60"
    # D: 24.000 MHz, H: 31.455 kHz, V: 59.915 Hz
    geometry 640 480 640 480 32
    timings 41666 37 12 33 10 74 2
    rgba 8/16,8/8,8/0,0/0
endmode
```

To change the video resolution and the timing the predefined settings from the video mode database could be used:

```
target# fbset 1024x768-75
target# fbset

mode "1024x768-75"
    # D: 84.005 MHz, H: 60.004 kHz, V: 75.005 Hz
    geometry 1024 768 1024 768 32
    timings 11904 229 23 28 1 124 3
    hsync high
    vsync high
    accel true
    rgba 0/0,0/0,0/0,0/0
endmode
```

The –move parameter could be used to change the visible picture position on the screen.

```
target# fbset -move up
```

The picture moving is achieved by changing the timing of hsync, vsync and the picture signal.

For a more detailed description take a look into the man page of **`fbset`** (on the TQ-CD).

7.2.3 Dual screen feature

Newer versions of the framebuffer driver are supporting the SM501 dual screen feature. This means that different pictures (with different resolutions and timings) could be displayed on the different interfaces (FP and CRT). A description could be found in the man page **`fb_sm501.4`**.

7.3 SRAM Driver

The **`SRAM_DRV`** driver is a simple device driver which can be used to access the on-board Static RAM (SRAM).

Hint: The SRAM driver could only be used on TQM5200 modules with on-board SRAM. There are some module variants which don't have SRAM. Make sure not to activate the driver on this variants during kernel configuration.

The driver prints a message during kernel boot time if it is loaded successfully:

```
target# dmesg | grep -i sram
SRAM_DRV: initialized
```

The device node has the major number of 32 and minor number 0. If it doesn't exist you have to create it before the driver could be used:

```
target# mknod /dev/sram c 32 0
target# ls -l /dev/sram
crw-r--r--    1 root     root      32,   0 Mar 13 16:43 /dev/sram
```

Now you could access the on-board SRAM over the special device file **`/dev/sram`**. There are two principal methods accessing the SRAM: with standard **`read()`** and **`write()`** system calls or with **`mmap()`**. For a full description of the supported system calls see the man page for the driver. You could find the man page in the Linux source tree (**`./linuxppc_2_4_devel/Documentation/man4/sram_drv.4`**) or as PDF file on the TQ-CD.

The test program **`sram`** could be used as example on how to access the SRAM.

Usage of **`sram`**:

```
target# ./sram
usage: ./sram read       offset len
       ./sram write      offset file
       ./sram mmap_read  offset len
       ./sram mmap_write offset file
```

Example of writing to the SRAM via the **`write()`** method:

```
target# ./sram write 0 en.txt
Writing 2069 bytes from offset 0x00000
         0   54 68 65 20  71 75 69 63  6B 20 62 72  6F 77 6E 20  | The quick brown
        10   64 6F 67 20  6A 75 6D 70  65 64 20 6F  76 65 72 20  | dog jumped over
       …
       800   6F 76 65 72  20 74 68 65  20 6C 61 7A  79 20 66 6F  | over the lazy fo
       810   78 2E 0D 0A  00                                     | x....
```

After booting the system, the content of the SRAM should be preserved (assuming, that the SRAM is buffered by a battery).

Example of reading from the SRAM via the **`mmap()`** method:

```
target# ./sram mmap_read 0 32
Reading 32 bytes from offset 0x00000
         0   54 68 65 20  71 75 69 63  6B 20 62 72  6F 77 6E 20  | The quick brown
        10   64 6F 67 20  6A 75 6D 70  65 64 20 6F  76 65 72 20  | dog jumped over
```

The test program **`sram`** could be compiled with this commands:

```
host# export CROSS_COMPILE=ppx_6xx-
host# ppx_6xx-gcc -O2 -s -o sram sram.c
```

7.4 PCAN Driver

A CAN driver for the MPC5200 internal CAN controller can be found on the DENX homepage [5]. Look for "PEAK CAN" (PCAN). The driver is also available on the TQ-CD: as compressed source tarball and as precompiled kernel module.

Please follow the instructions in **`README.peak-mpc5200`** to compile and/or use the PCAN driver.

Hint: Routing of the CAN signals is different on the STK52xx.100 and the STK52xx.200. So please make sure to use the right U-Boot configuration for the used baseboard. If you use an U-Boot which is configured for STK52xx.100 (CONFIG_STK52XX_REV100) on an STK52xx.200 baseboard the CAN interface does not work and vice versa.

7.4.1 PCAN on Boards with Rev. B MPC5200 CPU

On boards equipped with a MPC5200B CPU, please use the following PCAN driver version: **`peak-linux-driver-3.17-mpc5200_2007-05-22_tq.tar.gz`** (available on the TQ-CD)

This driver checks if it is running on an old MPC5200 CPU or on the newer Rev. B Version. If it runs on the MPC5200B, the external clock (SYS_XTAL_IN) is used as CAN clock source. On the old MPC5200 the IP bus clock is used.

Why are different clock sources used? This is because Freescale recommends to use the external clock (see Chapter 19.7.5 Clock System in the MPC5200 User's Manual), but on the old MPC5200 only the IP bus clock could be selected (because of a processor bug).

Please note, that different baud rate configuration values (BTR0BTR1) have to be used for boards equipped with MPC5200 and MPC5200B (see /Documentation/README-peack-mpc5200 in the pcan source tree).

7.5 USB Mass Storage Devices

An USB mass storage device with growing popularity is the USB stick. This chapter describes how USB sticks could be accessed in Linux.

Hint: If you want to access USB mass storage devices in Linux you must enable support for SCSI disks in the kernel configuration (see the chapter about the TQM5200 specific kernel configuration).

With the proc filesystem you could check if USB mass storage devices are supported in the current kernel:

```
target# cat bus/usb/drivers
         usbdevfs
         hub
 96-111: hiddev
         hid
         usb-storage
target# cat /proc/bus/usb/drivers
         usbdevfs
         hub
 96-111: hiddev
         hid
         usb-storage
target# cat /proc/scsi/
cat: /proc/scsi/: Is a directory
target# cat /proc/scsi/scsi
Attached devices: none
```

Now you could attach an USB stick (Hint: on the STK52xx only the lower USB connector could be used). When the USB stick is recognized, the Kernel prints a message on the console:

```
Vendor: JetFlash  Model: 32MB              Rev: 1.11
  Type:   Direct-Access                      ANSI SCSI revision: 02
Attached scsi removable disk sda at scsi0, channel 0, id 0, lun 0
SCSI device sda: 64512 512-byte hdwr sectors (33 MB)
sda: Write Protect is off
```

The USB stick is installed as removable disk **sda** by the SCSI subsystem. The disk could be accessed by the corresponding device node **/dev/sda**. If the SCSI device nodes don't exist you could create them with the **mknod** tool:

```
target# mknod /dev/sda  b 8 0
target# mknod /dev/sda1  b 8 1
target# mknod /dev/sda2  b 8 2
target# mknod /dev/sda3  b 8 3
target# mknod /dev/sda4  b 8 4
target# ll /dev/sda*
brw-r--r--    1 root     root       8,   0 May 17  2005 /dev/sda
brw-r--r--    1 root     root       8,   1 May 17  2005 /dev/sda1
brw-r--r--    1 root     root       8,   2 May 17  2005 /dev/sda2
brw-r--r--    1 root     root       8,   3 May 17  2005 /dev/sda3
brw-r--r--    1 root     root       8,   4 May 17  2005 /dev/sda4
```

With the proc interface some additional information about the USB stick could be displayed:

```
target# cat /proc/bus/usb/devices
T:  Bus=01 Lev=00 Prnt=00 Port=00 Cnt=00 Dev#=  1 Spd=12  MxCh= 1
B:  Alloc=  0/900 us ( 0%), #Int=  0, #Iso=  0
D:  Ver= 1.10 Cls=09(hub  ) Sub=00 Prot=00 MxPS= 8 #Cfgs=  1
P:  Vendor=0000 ProdID=0000 Rev= 0.00
S:  Product=USB OHCI Root Hub
S:  SerialNumber=f0001000
C:* #Ifs= 1 Cfg#= 1 Atr=40 MxPwr=  0mA
I:  If#= 0 Alt= 0 #EPs= 1 Cls=09(hub  ) Sub=00 Prot=00 Driver=hub
E:  Ad=81(I) Atr=03(Int.) MxPS=   2 Ivl=255ms
T:  Bus=01 Lev=01 Prnt=01 Port=00 Cnt=01 Dev#=  2 Spd=12  MxCh= 0
D:  Ver= 1.10 Cls=00(>ifc ) Sub=00 Prot=00 MxPS=64 #Cfgs=  1
P:  Vendor=0ea0 ProdID=6803 Rev= 1.00
S:  Manufacturer=USB
S:  Product=Solid state disk
S:  SerialNumber=403210023ED2C367
C:* #Ifs= 1 Cfg#= 1 Atr=80 MxPwr=100mA
I:  If#= 0 Alt= 0 #EPs= 3 Cls=08(stor.) Sub=06 Prot=50 Driver=usb-storage
E:  Ad=81(I) Atr=02(Bulk) MxPS=  64 Ivl=0ms
E:  Ad=02(O) Atr=02(Bulk) MxPS=  64 Ivl=0ms
E:  Ad=83(I) Atr=03(Int.) MxPS=   2 Ivl=1ms
target# cat /proc/scsi/scsi
Attached devices:
Host: scsi0 Channel: 00 Id: 00 Lun: 00
  Vendor: JetFlash Model: 32MB             Rev: 1.11
  Type:   Direct-Access                    ANSI SCSI revision: 02
target# cat /proc/scsi/usb-storage-0/0
   Host scsi0: usb-storage
       Vendor: USB
      Product: Solid state disk
Serial Number: 403210023ED2C367
     Protocol: Transparent SCSI
    Transport: Bulk
         GUID: 0ea06803403210023ed2c367
     Attached: Yes
```

The tool **scsi_info** also could be used to show some information about the USB stick:

```
target# scsi_info /dev/sda
SCSI_ID="0,0,0"
MODEL="JetFlash 32MB"
FW_REV="1.11"
```

Now we could check the partitions on the USB stick:

```
target# fdisk -l /dev/sda

Disk /dev/sda: 33 MB, 33030144 bytes
2 heads, 32 sectors/track, 1008 cylinders
Units = cylinders of 64 * 512 = 32768 bytes

   Device Boot    Start       End    Blocks   Id  System
/dev/sda1   *         1      1007     32208    4  FAT16 <32M
```

The example USB stick only has one FAT16 partition, which we could mount:

```
target# mount /dev/sda1 /mnt/usb/
target# mount
/dev/nfs on / type nfs (rw)
none on /proc type proc (rw)
usbdevfs on /proc/bus/usb type usbdevfs (rw)
/dev/sda1 on /mnt/usb type vfat (rw)
```

Now the mounted USB stick could be accessed as part of the root filesystem tree:

```
target# ll /mnt/usb/
total 16958
-rwxr--r--    1 root     root        782828 Mar 18  2005 kernel.img
drwxr--r--    5 root     root          2048 Dec 31  1979 microwindows
-rwxr--r--    1 root     root      16578134 Apr  6  2005 uRamdisk_tqm5200_16MB_ne
t-test_mkr.img
target# touch /mnt/usb/testfile
target# ll /mnt/usb/
total 16958
-rwxr--r--    1 root     root        782828 Mar 18  2005 kernel.img
drwxr--r--    5 root     root          2048 Dec 31  1979 microwindows
-rwxr--r--    1 root     root             0 Dec 31 19:20 testfile
-rwxr--r--    1 root     root      16578134 Apr  6  2005 uRamdisk_tqm5200_16MB_ne
t-test_mkr.img
```

Hint: Before removing the USB device you have to **umount** it, to write buffered data back to the device. Otherwise data on the USB stick could be corrupted.

```
target# umount /mnt/usb/
```

7.6 IDE Devices

The IDE devices on the TQM5200 are handled identically like on a x86 Linux host. Devices like ATA disk drives, CD-ROM drives and CompactFlash cards are supported.

The MPC5200 only supports one IDE channel (the primary channel). The master device on the primary channel is assigned to the device node /dev/hda, the slave device to /dev/hdb.

If the corresponding device nodes don't exist you could create them with the **mknod** tool:

```
target# mknod /dev/hda  b 3 0
target# mknod /dev/hda1  b 3 1
target# mknod /dev/hda2  b 3 2
target# mknod /dev/hda3  b 3 3
…
```

and

```
target# mknod /dev/hdb  b 3 64
target# mknod /dev/hdb1  b 3 65
target# mknod /dev/hdb2  b 3 66
target# mknod /dev/hdb3  b 3 67
…
```

During boot time the kernel prints some information about the attached IDE devices (in the example for a Toshiba CompactFlash card):

```
Probing IDE interface ide0...
hda: TOSHIBA THNCF128MBA, CFA DISK drive
ide0 at 0xf0003a60-0xf0003a67,0xf0003a5c on irq 45
hda: attached ide-disk driver.
hda: 250368 sectors (128 MB) w/2KiB Cache, CHS=978/8/32
Partition check:
 hda: hda1
```

In the following example two partitions are created on the CompactFlash card. A FAT16 msdos partition and a ext2 linux partition. Afterwards the filesystems are created and the partitions are mounted.

```
target# fdisk /dev/hda

Command (m for help): p

Disk /dev/hda: 128 MB, 128188416 bytes
8 heads, 32 sectors/track, 978 cylinders
Units = cylinders of 256 * 512 = 131072 bytes

   Device Boot    Start       End    Blocks   Id  System
/dev/hda1   *         1       977    125040    6  FAT16

Command (m for help): d
Selected partition 1

Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
Partition number (1-4): 1
First cylinder (1-978, default 1):
Using default value 1
Last cylinder or +size or +sizeM or +sizeK (1-978, default 978): +16M

Command (m for help): p

Disk /dev/hda: 128 MB, 128188416 bytes
8 heads, 32 sectors/track, 978 cylinders
Units = cylinders of 256 * 512 = 131072 bytes

   Device Boot    Start       End    Blocks   Id  System
/dev/hda1             1       123     15728   83  Linux

Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
Partition number (1-4): 2
First cylinder (124-978, default 124):
Using default value 124
Last cylinder or +size or +sizeM or +sizeK (124-978, default 978):
Using default value 978
```

```
Command (m for help): p

Disk /dev/hda: 128 MB, 128188416 bytes
8 heads, 32 sectors/track, 978 cylinders
Units = cylinders of 256 * 512 = 131072 bytes

   Device Boot    Start       End    Blocks   Id  System
/dev/hda1             1       123     15728   83  Linux
/dev/hda2           124       978    109440   83  Linux

Command (m for help): t
Partition number (1-4): 1
Hex code (type L to list codes): l

 0  Empty           1c  Hidden Win95 FA 70  DiskSecure Mult bb  Boot Wizard hid
 1  FAT12           1e  Hidden Win95 FA 75  PC/IX           be  Solaris boot
 2  XENIX root      24  NEC DOS         80  Old Minix       c1  DRDOS/sec (FAT-
 3  XENIX usr       39  Plan 9          81  Minix / old Lin c4  DRDOS/sec (FAT-
 4  FAT16 <32M      3c  PartitionMagic  82  Linux swap      c6  DRDOS/sec (FAT-
 5  Extended        40  Venix 80286     83  Linux           c7  Syrinx
 6  FAT16           41  PPC PReP Boot   84  OS/2 hidden C:  da  Non-FS data
 7  HPFS/NTFS       42  SFS             85  Linux extended  db  CP/M / CTOS / .
 8  AIX             4d  QNX4.x          86  NTFS volume set de  Dell Utility
 9  AIX bootable    4e  QNX4.x 2nd part 87  NTFS volume set df  BootIt
 a  OS/2 Boot Manag 4f  QNX4.x 3rd part 8e  Linux LVM       e1  DOS access
 b  Win95 FAT32     50  OnTrack DM      93  Amoeba          e3  DOS R/O
 c  Win95 FAT32 (LB 51  OnTrack DM6 Aux 94  Amoeba BBT      e4  SpeedStor
 e  Win95 FAT16 (LB 52  CP/M            9f  BSD/OS          eb  BeOS fs
 f  Win95 Ext'd (LB 53  OnTrack DM6 Aux a0  IBM Thinkpad hi ee  EFI GPT
10  OPUS            54  OnTrackDM6      a5  FreeBSD         ef  EFI (FAT-12/16/
11  Hidden FAT12    55  EZ-Drive        a6  OpenBSD         f0  Linux/PA-RISC b
12  Compaq diagnost 56  Golden Bow      a7  NeXTSTEP        f1  SpeedStor
14  Hidden FAT16 <3 5c  Priam Edisk     a8  Darwin UFS      f4  SpeedStor
16  Hidden FAT16    61  SpeedStor       a9  NetBSD          f2  DOS secondary
17  Hidden HPFS/NTF 63  GNU HURD or Sys ab  Darwin boot     fd  Linux raid auto
18  AST SmartSleep  64  Novell Netware  b7  BSDI fs         fe  LANstep
1b  Hidden Win95 FA 65  Novell Netware  b8  BSDI swap       ff  BBT
Hex code (type L to list codes): 4
Changed system type of partition 1 to 4 (FAT16 <32M)

Command (m for help): p

Disk /dev/hda: 128 MB, 128188416 bytes
8 heads, 32 sectors/track, 978 cylinders
Units = cylinders of 256 * 512 = 131072 bytes

   Device Boot    Start       End    Blocks   Id  System
/dev/hda1             1       123     15728    4  FAT16 <32M
/dev/hda2           124       978    109440   83  Linux

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.

WARNING: If you have created or modified any DOS 6.x
partitions, please see the fdisk manual page for additional
information.
Syncing disks.
target#
target# mke2fs /dev/hda2
mke2fs 1.27 (8-Mar-2002)
```

```
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
27440 inodes, 109440 blocks
5472 blocks (5.00%) reserved for the super user
First data block=1
14 block groups
8192 blocks per group, 8192 fragments per group
1960 inodes per group
Superblock backups stored on blocks:
        8193, 24577, 40961, 57345, 73729

Writing inode tables: done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 33 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
target# mkfs.vfat /dev/hda1
mkfs.vfat 2.8 (28 Feb 2001)
target#
target# mount
/dev/nfs on / type nfs (rw)
none on /proc type proc (rw)
usbdevfs on /proc/bus/usb type usbdevfs (rw)
target# mount /dev/hda1 /mnt/hda1
target# mount /dev/hda2 /mnt/hda2
target# mount
/dev/nfs on / type nfs (rw)
none on /proc type proc (rw)
usbdevfs on /proc/bus/usb type usbdevfs (rw)
/dev/hda1 on /mnt/hda1 type vfat (rw)
/dev/hda2 on /mnt/hda2 type ext2 (rw)
target# ls -l /mnt/hda1
total 0
target# ls -l /mnt/hda2
total 12
drwx------    2 root     root        12288 Dec 31 19:06 lost+found
target# touch /mnt/hda1 tesetfile
target# touch /mnt/hda2 tesetfile
target# ls -l /mnt/hda1
total 0
target# ls
tesetfile
target# rm tesetfile
target# touch /mnt/hda1/testfile
target# touch /mnt/hda2/testfile
target# ls -l /mnt/hda1
total 0
-rwxr--r--    1 root     root            0 Dec 31 19:29 testfile
target# ls -l /mnt/hda2
total 12
drwx------    2 root     root        12288 Dec 31 19:06 lost+found
-rw-r--r--    1 root     root            0 Dec 31 19:29 testfile
target# umount /mnt/hda1
target# umount /mnt/hda2
```

Hint: Don't forget to **umount** the mounted IDE devices, to write buffered data back to the device. Otherwise data written to the IDE device could be lost.

7.7 A Microwindows Demo

This chapter describes the installation of a little Microwindows demo.

Hint: You have to use a TQM5200 module variant with on-board graphic controller and the used Linux kernel must be compiled with framebuffer driver support.

The easiest way to use the demo is, to install it on the NFS filesystem mounted by the target. To do so, copy the tar archive **microwindows-demo_tqm5200.tar.gz** from the TQ-CD to the NFS filesystem and unpack it.

On the host:

```
host# cp <path on cd>/microwindows-demo_tqm5200.tar.gz /opt/eldk/ppx_6xx/home/
```

On the target:

```
target# cd /home/
target# tar -xzvf microwindows-demo_tqm5200.tar.gz
microwindows/
microwindows/bin/
microwindows/bin/tuxchesss
microwindows/bin/nterm
microwindows/bin/nano-Xs
microwindows/bin/nanowms
microwindows/bin/nano-X
microwindows/bin/nanowm
microwindows/bin/nterms
microwindows/bin/nxterm
microwindows/bin/tuxchess
microwindows/bin/nxterms
microwindows/lib/
microwindows/lib/libmwinlib.so
microwindows/lib/libmwin.so
microwindows/lib/libmwfonts.so
microwindows/lib/libmwimages.so
microwindows/lib/libnano-X.so
microwindows/lib/libmwengine.so
microwindows/lib/libmwdrivers.so
microwindows/tuxchess/
microwindows/tuxchess/images/
microwindows/tuxchess/images/README.images
microwindows/tuxchess/images/b_b.gif
microwindows/tuxchess/images/b_k.gif
microwindows/tuxchess/images/b_n.gif
microwindows/tuxchess/images/b_p.gif
microwindows/tuxchess/images/b_q.gif
microwindows/tuxchess/images/b_r.gif
microwindows/tuxchess/images/w_b.gif
microwindows/tuxchess/images/w_k.gif
microwindows/tuxchess/images/w_n.gif
microwindows/tuxchess/images/w_p.gif
microwindows/tuxchess/images/w_q.gif
microwindows/tuxchess/images/w_r.gif
microwindows/tuxchess/images/board.gif
startm
startms
```

If the device node for the PS/2 mouse doesn't exist, you could create it with the **mknod** tool:

```
target# mknod /dev/psaux c 10 1
```

There are two versions of the demo files. One version is created to use dynamic libraries and the other version uses static libraries. The files compiled with static libraries have a trailing “s” in the name.

To start the version with dynamic libraries run **./startm**. To run the version with static libraries use **./startms**. The script starts the nano-X server, a windowmanager, a console window and a chess game (tuxchess). Use the PS/2 mouse and the PS/2 keyboard connected to the STK52xx to control the demo.

You could quit the demo by pressing the left and the right mouse button simultaneously.

On the ELDK target NFS filesystem you could find an already precompiled (minesweeper) demo, too. It is started with **/usr/bin/mine**. Because this demo is not optimized for the TQM5200 the graphic output is a little bit “bluish”.

The microwindows sources could be found on the ELDK source CD as a source RPM (see [8]).

7.8 Java Support

For the TQM5200 there exist two Implementations of Java Virtual Machines:

- Blackdown J2RE1.3.1-02B-FCS [11]
- JamaicaVM J2SDK1.4.2 [12]

7.8.1 Blackdown

This chapter describes the installation of the Blackdown Java 2 runtime environment for the TQM5200 Module.

On the TQ-CD there is a tar archive (**blackdown_2006-01-23.tar**) which contains the Blackdown runtime environment, a document with installation instructions and some basic benchmarks. First Install the tar archive on your host:

```
host# tar -xvf blackdown_2006-01-23.tar
CaffeineMark.log
Dhrystone.log
Whetstone.log
Dhrystone.class
DhrystoneConstants.class
GlobalVariables.class
Record_Type.class
Whetstone.class
j2re-1.3.1-02b-FCS-linux-ppc.bin
caffeine.tgz
Blackdown_jvm_usage.txt
```

Then follow the instructions in the document **Blackdown_jvm_usage.txt.**

7.8.1.1 Graphical Benchmark

To run the graphical benchmark “CaffeineMark 3.0” use the tar archive **blackdown_jvm_2006-05-19.zip** on the TQ-CD and follow the instructions in the included file **“Blackdown_jvm_usage.txt”.**

LIN5200.SWM.105.doc

Prerequisite:

To run the graphical benchmark a running X11 system is required. For example the port of the "ubuntu" distribution for the TQM5200 (see separate chapter).

Results:

For the test the following hardware was used:

- STK52xx.200
- TQM5200.200 (128MB RAM, 64MB Flash, 132MHz IPB, 66MHz PCI)

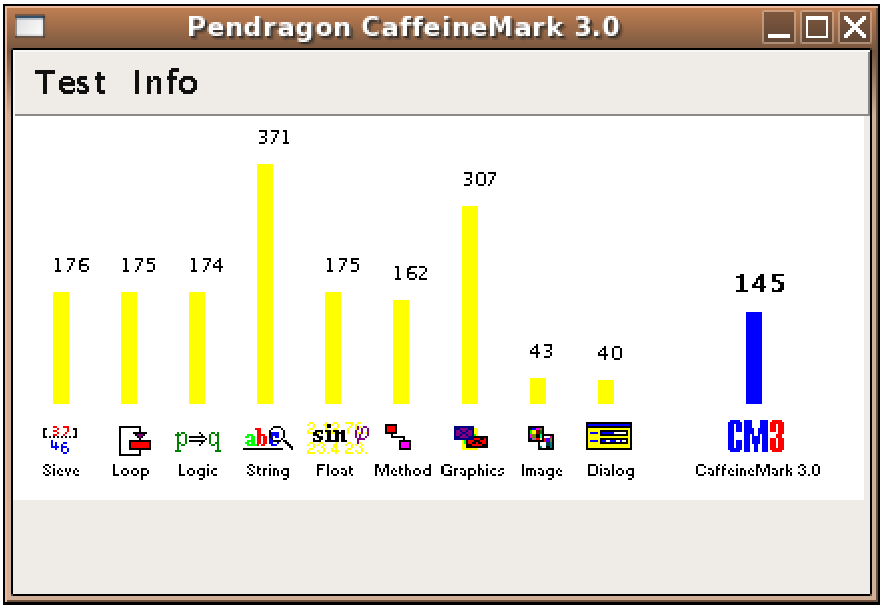

Figure 1: CaffeineMark 3.0 results

7.8.2 JamaicaVM

This chapter describes the installation of the JamaicaVM evaluation software for the TQM5200 Module.

First install the tar archive **`Jamaica-2.8-0-linux-ppc-without-awt-linux-linux-ppc603.tar.gz`** from the TQ-CD on your host:

```
host# tar -xvf Jamaica-2.8-0-linux-ppc-without-awt-linux-linux-ppc603.tar.gz
Jamaica.install
Jamaica.license
Jamaica.readme
Jamaica.remove
Jamaica.ss
```

Then extract the **`Jamaica.ss`** archive:

```
host# tar -xpf Jamaica.ss
tar: Removing leading `/' from member names
tar: A lone zero block at 403460
```

The JamiacaVM Toolset is now installed on your host. To be able to use the tools, set your PATH variable accordingly:

```
host# export JAMAICA=`pwd`/usr/local/jamaica-2.8-0
host# export PATH=$JAMAICA/bin:$PATH
```

7.8.2.1 HelloWorld example

Hint: To be able to compile the Example you need a running Java compiler on your host. For example the **javac** compiler from the j2sdk1.4.2 packet from Sun [13]. Installation:

```
host# ./j2sdk-1_4_2_10-linux-i586.bin
[...]
  inflating: j2sdk1.4.2_10/demo/jfc/Java2D/README.txt
  inflating: j2sdk1.4.2_10/demo/jfc/Java2D/Java2Demo.html
  inflating: j2sdk1.4.2_10/demo/jfc/Java2D/Java2Demo.jar
   creating: j2sdk1.4.2_10/demo/jfc/Font2DTest/
   creating: j2sdk1.4.2_10/demo/jfc/Font2DTest/src/
  inflating: j2sdk1.4.2_10/demo/jfc/Font2DTest/src/Font2DTest.java
[...]
host# export PATH=`pwd`/j2sdk1.4.2_10/bin/:$PATH
```

Hint: The ELDK (or another cross toolchain) for the MPC5200 has to be installed on your host, to be able to generate a HelloWorld application with Jamaica. The ELDK installation path in the file **jamaica-2.8-0/etc/jamaica.conf** must match your configuration. Change it if necessary.

Hint: If you have installed the JamaicaVM in a directory other than **/usr/local** you have to adjust the variable “JAMAICA” in the **Makefile**.

Compiling the bytecode from the HelloWorld source:

```
host# cd usr/local/jamaica-2.8-0/target/linux-gnu-ppc603/examples/HelloWorld/
host# make classes/HelloWorld.class
mkdir -p classes
javac -classpath classes -bootclasspath /home/mkr/jamaica/usr/local/jamaica-2.8-
0/classes -sourcepath src -d classes src/HelloWorld.java
```

This generates the Java bytecode for the HelloWorld example. The bytecode could now be interpreted by a Java Virtual Machine.

Running the example on the host:

```
host# jamaicavm -classpath classes HelloWorld
              Hello        World!
           Hello       World!
        Hello     World!
     Hello    World!
   Hello   World!
 Hello  World!
Hello World!
Hello World!
Hello World!
Hello World!
 Hello  World!
   Hello   World!
     Hello    World!
        Hello     World!
           Hello       World!
              Hello        World!
                 Hello         World!
                    Hello           World!
                       Hello            World!
```


LIN5200.SWM.105.doc

```
                                Hello              World!
                                  Hello              World!
                                   Hello               World!
                                   Hello               World!
                                   Hello               World!
                                   Hello               World!
                                  Hello              World!
                                 Hello             World!
                               Hello             World!
                             Hello           World!
                          Hello         World!
```

Building the example for the target:

The Jamaica Builder integrates the JVM and all necessary classes into one application und does a couple of optimizations. This approach leads to much faster applications than interpreting the Java class code by a standalone JVM. For Details see the JamaicaVM documentation [12].

```
host# make
[…]
Reading configuration from '/home/mkr/jamaica/usr/local/jamaica-2.8-
0/etc/jamaica.conf'...
Jamaica Builder Tool 2.8 Release 0 (User: krause, Expires: 2007.01.25)
Generating code for target 'linux-gnu-ppc603', optimisation 'speed'
 + tmp/HelloWorld_micro__.c
 + tmp/HelloWorld_micro__.h
Class file compaction gain: 76.40517% (2261232 ==> 533534)
 * C compiling tmp/HelloWorld_micro__.c
linking
host# file HelloWorld
HelloWorld: ELF 32-bit MSB executable, PowerPC or cisco 4500, version 1 (SYSV), for
GNU/Linux 2.2.5, dynamically linked (uses shared libs), not stripped
```

The application **`HelloWorld`** now can be copied to the target.

7.8.2.2 Jamaica JVM for the target

Apart from building an optimized image consisting of both, the JVM and the actual Java application - like described above – a standalone JVM could be installed on the target, interpreting Java class code files. This would be handy during application development, because it is easier to run the class code directly, than building the optimized target application for every test run.

The target JVM is located in the file **`HelloWorld-Jamaica-2.8-Linux-PPC-6xx.zip`** on the TQ-CD. Unzip it and copy the file **`jamaicavm_ppc6xx`** to the target.

Now Java class code (**`HelloWorld.class`**) could be run by the JVM on the target:

```
target# ./jamaicavm_ppc6xx HelloWorld
              Hello        World!
            Hello       World!
          Hello     World!
       Hello    World!
    Hello   World!
  Hello  World!
Hello World!
Hello World!
Hello World!
Hello World!
  Hello  World!
    Hello   World!
```

```
Hello    World!
   Hello     World!
      Hello       World!
         Hello        World!
            Hello         World!
               Hello           World!
                  Hello            World!
                    Hello             World!
                      Hello             World!
                       Hello               World!
                       Hello               World!
                       Hello               World!
                       Hello               World!
                      Hello             World!
                    Hello             World!
                  Hello            World!
               Hello           World!
            Hello         World!
```

7.9 Ubuntu with X11 support and graphical desktop

For the TQM5200 there exists a port of the Ubuntu distribution. This includes for example a X11 server, a graphical desktop (GNOME) and an accelerated graphic driver for the SM501.

Because the whole installation of Ubuntu is very big (over 3 Gbyte) it is not contained on the TQ-CD. Please contact TQ-Components directly if you are interested in an sample installation (hard disk image).

Hint: For running X11 on the TQM5200 at least 64 Mbyte SDRAM are required.

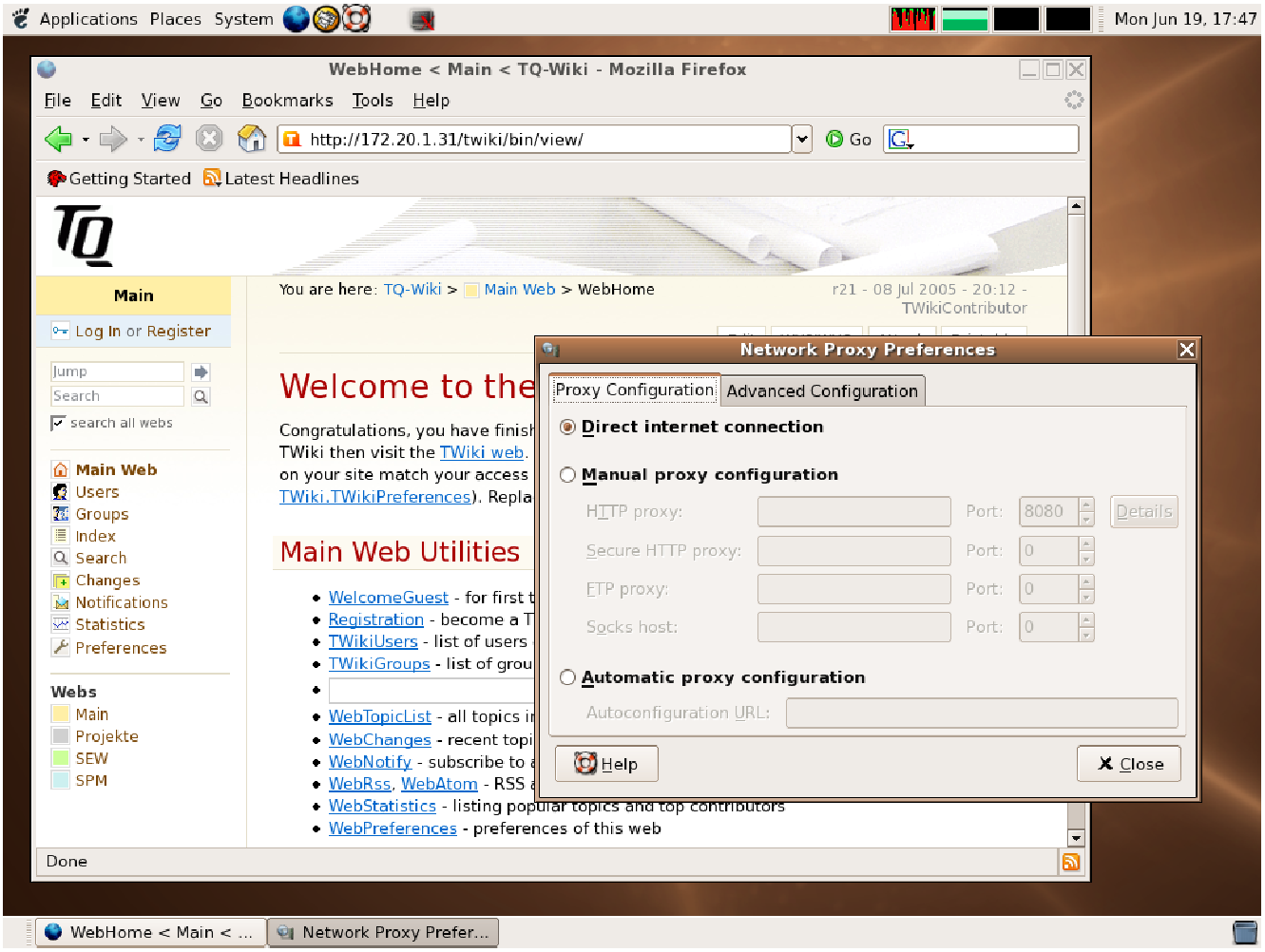

Figure 2: Screenshot of the GNOME desktop running on the TQM5200

8. Specification

8.1 U-Boot

The following configurations are possible for the U-Boot on the TQM5200:

Module Variant	Compiler Call	Description
TQM5200-AA	`make TQM5200_AA_config`	Obsolete
TQM5200-AB	`make TQM5200_AB_config`	Obsolete
TQM5200-AC	`make TQM5200_AC_config`	Obsolete
TQM5200-AD	`make TQM5200_AB_config`	Obsolete
TQM5200	`make TQM5200_auto_config`	Automatic HW detection, should run on all modules. This is the preferred configuration for STK52xx baseboards
TQM5200	`make TQM5200_config`	Used with newer U-Boot versions (replaces **`TQM5200_auto_config`**)
TQM5200S;TQM5200 Revision 214 and above (with MPC5200B cpu)	`make TQM5200_B_config`	For TQM5200S and TQM5200 with MPC5200B CPU. Attention: new Flash map compared to the old TQM5200!
TQM5200S;TQM5200 Revision 214 and above (with MPC5200B cpu)	`Make TQM5200_B_HIGHBOOT_config`	U-Boot for high-boot configuration (linked for address 0xFFF00000)
TB5200	`make TB5200_config`	Configuration for the TB5200 baseboard
TB5200 Rev. 300	`make TB5200_B_config`	Configuration for the TB5200 Rev. 300 baseboard (with MPC5200B cpu)

If the U-Boot is to be adapted individually, then the file **`include/configs/TQM5200.h`** (or the appropriate configuration file for the used baseboard) in the U-Boot source tree must be modified and the U-Boot must be compiled afresh. The description of the U-Boot configuration is to be found in [4].

8.1.1 Address Map

The address map (after initialization by the U-Boot) is configured as illustrated below:

8.1.1.1 TQM5200-AA:

Address Range	Chip Select	Device	Description
0x00000000 - 0x00FFFFFF	MEM_CS0	SDRAM	16 MByte
0xF0000000 - 0xF0003FFF	-	MBAR	MPC5200 Internal Register Memory Map
0xF0004000 - 0xF000BFFF	-	On-chip SRAM	16 kByte MPC5200 Internal
0xFC000000 - 0xFC3FFFFF	LP_CS0# (BOOT_CS)	Flash	4 MByte

8.1.1.2 TQM5200-AB / TQM5200-AD:

Address Range	Chip Select	Device	Description
0x00000000 - 0x03FFFFFF	MEM_CS0	SDRAM	64 MByte
0xE0000000 - 0xE07FFFFF	LP_CS1#	SM501	On-chip Graphics Memory
0xE3E00000 - 0xE3DFFFFF	LP_CS1#	SM501	Memory Mapped I/O Space
0xE5000000 - 0xE507FFFF	LP_CS2#	SRAM	512 kByte
0xF0000000 - 0xF0003FFF	-	MBAR	MPC5200 Internal Register Memory Map
0xF0004000 - 0xF000BFFF	-	On-chip SRAM	16 kByte MPC5200 Internal
0xFC000000 - 0xFDFFFFFF	LP_CS0# (BOOT_CS)	Flash	32 Mbyte

8.1.1.3 TQM5200-AC:

Address Range	Chip Select	Device	Description
0x00000000 - 0x07FFFFFF	MEM_CS0	SDRAM	128 Mbyte
0xE0000000 - 0xE07FFFFF	LP_CS1#	SM501	On-chip Graphics Memory
0xE3E00000 - 0xE3DFFFFF	LP_CS1#	SM501	Memory Mapped I/O Space
0xF0000000 - 0xF0003FFF	-	MBAR	MPC5200 Internal Register Memory Map

0xF0004000 - 0xF000BFFF	-	On-chip SRAM	16 kByte MPC5200 Internal
0xFC000000 - 0xFC3FFFFF	LP_CS0# (BOOT_CS)	Flash	4 Mbyte

8.1.1.4 TQM5200S-AA

Address Range	Chip Select	Device	Description
0x00000000 - 0x03FFFFFF	MEM_CS0	SDRAM	64 MByte
0xE5000000 - 0xE507FFFF	LP_CS2#	SRAM	512 kByte
0xF0000000 - 0xF0003FFF	MBAR	MPC5200	Internal Register Memory Map
0xF0004000 - 0xF000BFFF	On-chip	SRAM	16 kByte MPC5200 Internal
0xFC000000 - 0xFDFFFFFF	LP_CS0# (BOOT_CS)	Flash	32 Mbyte

8.1.1.5 TQM5200S-AB

Address Range	Chip Select	Device	Description
0x00000000 - 0x03FFFFFF	MEM_CS0	SDRAM	64 MByte
0xE5000000 - 0xE507FFFF	LP_CS2#	SRAM	512 kByte
0xF0000000 - 0xF0003FFF	MBAR	MPC5200	Internal Register Memory Map
0xFC000000 - 0xFCFFFFFF	LP_CS0# (BOOT_CS)	Flash	16 Mbyte

The U-Boot command **`reginfo`** displays the configuration of the CS registers at run-time.

8.1.2 Sources

The sources can be downloaded from the DENX git server [10]. As of date, not all modifications, which TQ has undertaken on the U-Boot for the TQM5200, have flown back into the git archive, but most of them already have. TQ provides patches for the missing parts (ask for them).

8.2 Linux

The U-Boot is often used in connection with embedded Linux.

A detailed documentation on embedded Linux is to be found in the DULG (DENX U-Boot Linux Guide): http://www.denx.de/twiki/bin/view/DULG/Manual.

The current kernel version, on which this manual is based, is 2.4.25

8.2.1 Sources

The Sources can be downloaded from the git server of the company DENX Software Engineering free of cost under the module name: **`linuxppc_2_4_devel`**. See [5] and [10].

As of date, not all modifications, which TQ has undertaken on the kernel for the TQM5200, have flown back into the git version from DENX, but most of them already have. TQ provides patches for the missing parts (ask for them).

The sources for the examples described in this manual can be found on the TQ-CD supplied with the STK52xx Starterkit.

8.2.2 Flash Memory Map

The Flash allocation used by the Linux kernel is illustrated below. If Linux is not used, then the complete Flash can be used starting after the Environment sector(s).

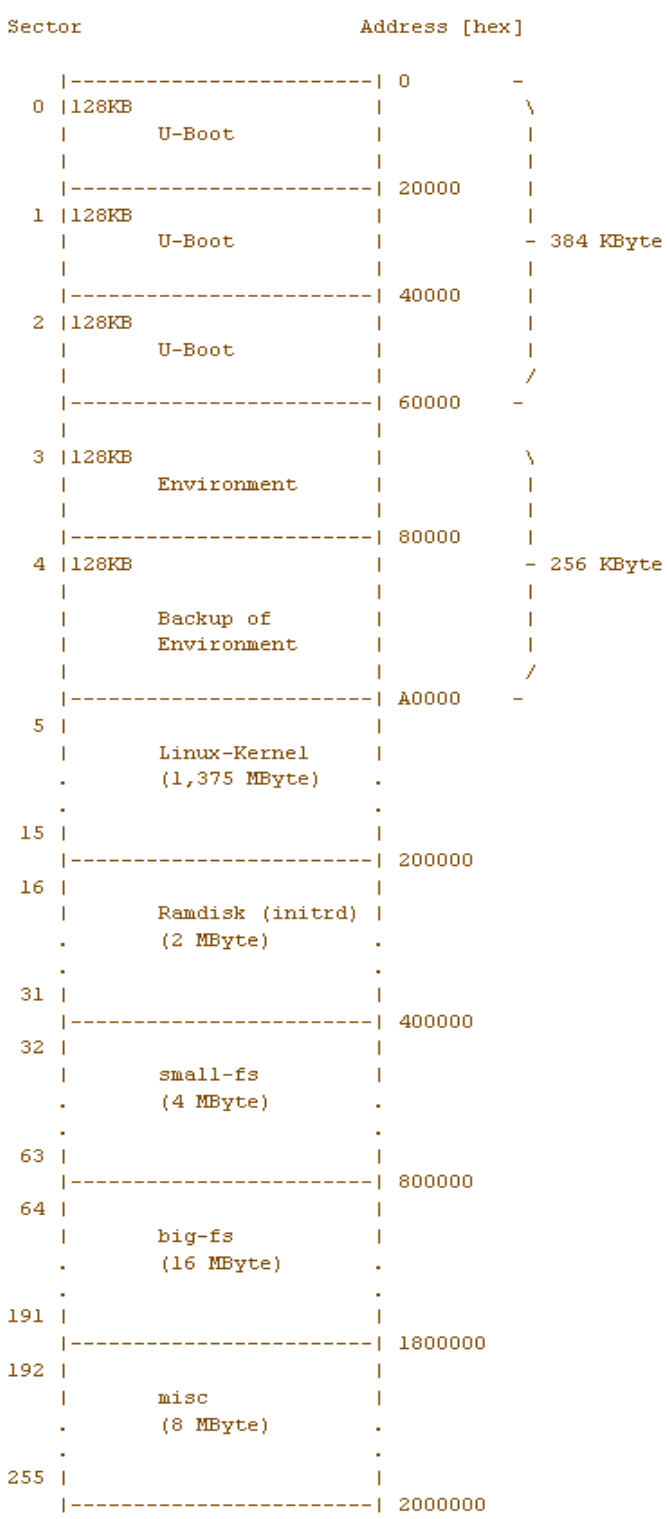

Figure 3: Flashmap for TQM5200 (Rev 200 - Rev 213)

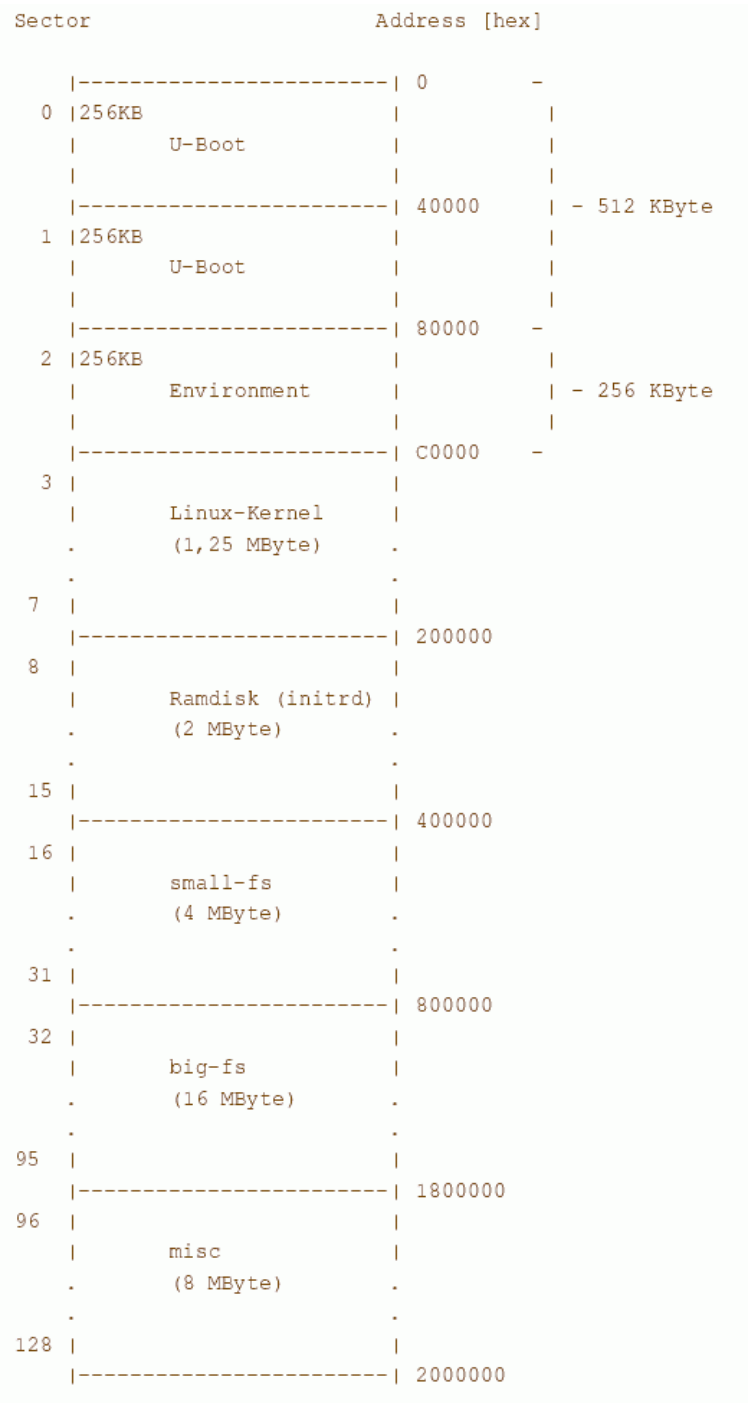

Figure 4: Flashmap for TQM5200 „Rev B“ (Rev 214 and above) and TQM520S

The Flash allocation is the same for all Flash variants. The figure shows the allocation for 32 MByte Flash (TQM5200-AB). With smaller Flash sizes the appropriate part in the higher order is omitted accordingly. The used flash devices on TQM5200-AA and TQM5200-AB are so called boot sector flashes. This flashes have some smaller sectors at the beginning. Thus the sector numbers shown above do not correspondent - but the shown addresses do.

The specified addresses are to be understood as offsets to the Flash base addresses. The base address is currently 0xFC000000.

! note !	The Flash allocation has changed with respect to earlier versions of the U-Boot. The U-Boot now occupies the first 3 sectors in the Flash, instead of the first 2 in earlier versions. Thus the location of the Environment has shifted by one sector. Also the start address of the Linux Kernel has shifted by one sector.

8.3 ELDK

The Embedded Linux Development Kit (**ELDK**) includes the GNU cross development tools, such as the compilers, binutils, gdb, etc., and a number of pre-built target tools and libraries necessary to provide some functionality on the target system.

It is provided for free with full source code, including all patches, extensions, programs and scripts used to build the tools.

Packaging and installation is based on the RPM package manager.

The ELDK is licensed under the GPL [9] by DENX Software Engineering [5].

Currently, the version 3.1.1 (ppc-2005-06-07) of the ELDK is used.

For recent versions look at [8].

8.4 Literature

[1] DULG (DENX U-Boot and Linux Guide): http://www.denx.de/twiki/bin/view/DULG/Manual

[2] TLDP (The Linux Documentation Project): http://www.tldp.org.

[3] Linux kernel frame-buffer documentation: directory `./Documentation/fb` in the Linux kernel source tree.

[4] U-Boot documentation: file README and directory /doc in the U-Boot source tree

[5] DENX Software Engineering homepage: http://www.denx.de

[6] “Das U-Boot - Universal Bootloader”: http://sourceforge.net/projects/u-boot

[7] Linux kernel documentation: directory `./Documentation` in the Linux kernel source tree

[8] ELDK Availability: http://www.denx.de/twiki/bin/view/DULG/ELDKAvailability

[9] The GNU General Public License (GPL): http://www.gnu.org/licenses/gpl.html

[10] git access for DENX Source Code: http://source.denx.net/cgi-bin/gitweb.cgi

[11] Blackdown Java Linux Homepage: http://www.blackdown.org/

[12] JamaicaVM Homepage: http://www.aicas.com

[13] Java 2 Platform from Sun Microsystems: http://java.sun.com/j2se/1.4.2/download.html

Notes